Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

A Comparative Performance & Metadata Study of Open Table Formats: Iceberg vs Delta vs Hudi at Scale

Siddhartha Parimi
Dell Technologies, USA
Corresponding Author: Siddhartha Parimi, **E-mail**: parimi.sid.work@gmail.com

| ABSTRACT

The rapid adoption of open table formats has fundamentally transformed modern data engineering by enabling ACID transactions, schema evolution, and time travel capabilities on cloud object storage systems. Apache Iceberg, Delta Lake, and Apache Hudi represent the three dominant solutions that have emerged to address traditional data lake limitations, including a lack of transactional guarantees, concurrent write challenges, and metadata management inefficiencies. This evaluation conducts empirical benchmarking across terabyte-scale datasets to compare these formats across critical dimensions, including metadata scalability, transaction isolation guarantees, concurrent write handling, compaction strategies, streaming consistency semantics, and cross-engine interoperability. Testing scenarios encompass bulk ingestion throughput, incremental write latency, selective query performance, time travel operations, schema evolution capabilities, and maintenance overhead under varying concurrency levels. Results reveal that Iceberg excels in read-heavy analytics workloads with superior query planning efficiency and cross-engine portability, Delta Lake demonstrates operational simplicity with strong Spark integration and the highest bulk write throughput, while Hudi offers flexible write-read tradeoffs through dual table types optimized for streaming upserts. Format convergence trends indicate rapid feature adoption across competing implementations, reducing vendor lock-in risks and enabling organizations to select formats based on specific workload characteristics rather than seeking universally optimal solutions. The article establishes quantitative foundations for practitioners navigating table format selection as lakehouse architectures become the dominant paradigm for enterprise data platforms, with direct implications for infrastructure costs, operational complexity, and analytical performance at scale.

| KEYWORDS

Lakehouse Architecture, Table Format Comparison, Metadata Management, Transaction Isolation, Distributed Data Systems

| ARTICLE INFORMATION

ACCEPTED: 20 December 2025 **PUBLISHED:** 29 December 2025 **DOI:** 10.32996/jcsts.2025.7.12.56

1. Introduction

The current data environment has undergone a paradigm shift due to the introduction of lakehouse architectures that consolidate data warehousing and data lake functionality into a single architecture. Lakehouse methodology is a performance and structure-based approach that integrates the performance and structure of data warehouses with the flexibility and cost-effectiveness of data lakes and allows organizations to store all their data at a single location and to support various analytical workloads such as business intelligence and machine learning [1]. This architectural evolution has been driven by the limitations of traditional two-tier architectures, where data lakes stored raw data separately from data warehouses, creating complexity, data duplication, and reliability challenges that hindered analytical capabilities.

Open table formats have emerged as the foundational technology enabling lakehouse implementations by providing ACID transaction guarantees, schema evolution, time travel capabilities, and metadata management on top of cloud object storage systems. Apache Iceberg, Delta Lake, and Apache Hudi represent the three dominant open table formats that have gained significant traction across enterprise organizations. Microsoft Azure documentation emphasizes that the lakehouse paradigm

delivers reliability and performance on data lakes through these open formats, enabling organizations to eliminate data silos and reduce infrastructure complexity while maintaining enterprise-grade data management capabilities [5]. Although these formats have been widely adopted and are increasingly of interest to various industries, there are few extensive empirical comparisons of the formats under controlled and large-scale environments, both in the academic literature and in industry publications.

In this research, the critical gap can be bridged by performing rigorous performance tests on terabyte-scale datasets, the metadata scalability, the scalability of transaction isolation, cross-engine interoperability, and cross-engine concurrent write handling. Our methodology employs controlled benchmarking scenarios that simulate real-world production workloads, including bulk ingestion, streaming writes, selective queries, and time travel operations. The findings provide quantitative insights into architectural tradeoffs inherent in each format and establish a reference framework for organizations selecting long-term table format strategies as they transition to lakehouse platforms.

2. Background and Architectural Foundations

The evolution of table formats represents a fundamental shift in how organizations approach large-scale data management on cloud infrastructure. Traditional data lakes built on file formats like Parquet and ORC lacked essential database features, including atomic operations, concurrent write support, and schema evolution capabilities. RCFile introduced columnar storage optimizations for MapReduce-based systems, demonstrating how data placement structures significantly impact query performance through improved compression ratios and reduced I/O operations during analytical processing [3]. These early innovations established the foundation for modern table formats by proving that metadata-driven approaches could dramatically enhance data lake performance without sacrificing the flexibility of object storage systems.

Apache Iceberg addresses metadata scalability challenges through its three-tier hierarchical structure consisting of metadata files, manifest lists, and manifest files that track data file locations and statistics. The technical documentation of Oracle points out that the architecture of Iceberg provides efficient pruning of partitions, query planning in that it keeps column-level statistics and supports the hidden schemes of partitioning, which facilitates evolution of partitions without rewriting of data [2]. The format uses optimistic concurrency control with snapshot isolation guarantees, such that parallel readers and writers can co-exist without conflicting reads or writes and have consistent views of table data. The storage-agnostic design of Iceberg does not provide explicit specifications of a particular storage system or compute engine, allowing wide adoption across a wide range of different analytical engines, such as Apache Spark, Apache Flink, Trino, and cloud-native query engines.

Delta Lake implements a transaction log architecture where all table modifications are recorded as ordered, immutable entries in a centralized log stored alongside table data. The format documentation describes how Delta Lake provides ACID guarantees through this log-based approach, with each commit receiving a monotonically increasing version number that establishes a total ordering of all table operations [10]. Checkpointing mechanisms periodically consolidate the transaction log to prevent unbounded growth and maintain efficient query planning performance. The format integrates deeply with Apache Spark's execution engine and provides features including deletion vectors for efficient update operations, liquid clustering for automatic data optimization, and UniForm capability enabling dual-format reads where tables written as Delta can be consumed as Iceberg tables by external engines.

Apache Hudi distinguishes itself through its flexible architecture, offering two distinct table types that enable workload-specific optimizations. Copy-on-Write tables rewrite entire data files during updates to maintain optimal read performance, while Merge-on-Read tables append changes to log files for lower write latency at the cost of increased read complexity. The Hudi documentation explains that this dual-mode approach allows organizations to tune their tables based on read-write ratio requirements, with Merge-on-Read excelling for streaming ingestion scenarios and Copy-on-Write preferred for analytical workloads prioritizing query performance [9]. The timeline service offered by Hudi has a global audit trail of table operations, which facilitates incremental processing patterns and supports efficient upsert operations with record-level index structures. The format offers configurable compaction policies that compromise write performance with storage efficiency and read latency, allowing practitioners to exercise control over the operational properties.

The convergence of these formats reflects broader trends in distributed systems design, where competing approaches gradually adopt successful features from one another. MapReduce systems evolved from simple batch processing frameworks into sophisticated platforms supporting iterative algorithms and interactive queries through architectural innovations in metadata management and query optimization [4]. Similarly, modern table formats continue evolving by incorporating complementary capabilities while maintaining their core architectural philosophies, creating an ecosystem where format selection depends increasingly on workload characteristics and organizational constraints rather than absolute technical superiority.

Format	Metadata Structure	Transaction Model	Storage Approach	Primary Design Goal
Apache Iceberg	Three-tier hierarchy (metadata files, manifest lists, manifests)	Optimistic concurrency with snapshot isolation	Complete separation of metadata and data	Read optimization and cross-engine portability
Delta Lake	Centralized append-only transaction log with JSON entries	Linearizable log-based commits with checkpointing	The transaction log is stored alongside the data	Operational simplicity and Spark integration
Apache Hudi	Timeline service with embedded metadata	File-level conflict detection with dual modes	Metadata embedded within data layout	Write flexibility and streaming optimization

Table 1: Architectural Characteristics of Open Table Formats [3, 4]

3. Methodology and Experimental Design

The experimental methodology employed controlled benchmarking across terabyte-scale synthetic datasets designed to represent common production workload patterns observed in enterprise data platforms. The benchmark environment utilized a dedicated compute cluster with specifications chosen to eliminate resource bottlenecks that might obscure format-specific performance characteristics. Dataset construction followed principles established in distributed database research, where synthetic data generation enables reproducible experiments while capturing essential characteristics of real-world workloads, including skewed distributions, temporal patterns, and high-cardinality dimensions.

The primary evaluation dataset consisted of five terabytes of e-commerce clickstream events with fifty billion individual records, incorporating high-cardinality attributes representing user identifiers, session identifiers, and product catalog references. Temporal partitioning by date and hour created thousands of leaf partitions reflecting typical production table structures where data arrives continuously, and queries target specific time ranges. A secondary dataset modeled industrial IoT sensor telemetry with three terabytes of time-series measurements generated by simulated sensor networks, while a financial transaction dataset provided two terabytes of structured records with monetary values and categorical attributes. These datasets enabled comprehensive evaluation across diverse access patterns, including sequential scans, selective queries with complex predicates, and point lookups requiring efficient metadata filtering.

Benchmark scenarios covered eight critical operational categories representing the full lifecycle of production table management. Bulk ingestion testing measured initial data loading performance and metadata creation overhead under varying parallelism levels from twenty to two hundred concurrent writers. Incremental write scenarios simulated streaming pipelines with micro-batches arriving at regular intervals, evaluating steady-state write latency and throughput characteristics. Concurrent write testing assessed transaction isolation and conflict resolution mechanisms by executing overlapping writes to shared partition ranges, measuring conflict rates and retry behavior under increasing concurrency pressure. Read performance evaluation included full table scans measuring raw throughput, selective queries testing predicate pushdown efficiency, and time travel operations accessing historical snapshots to evaluate snapshot resolution overhead.

Schema evolution testing validated each format's capabilities for adding columns, promoting data types, and evolving partition specifications, measuring both operation latency and impact on concurrent read-write workloads. Compaction testing addressed the small file problem inherent in streaming ingestion by consolidating accumulated micro-batch files and measuring optimization throughput alongside effects on concurrent operations. Cross-engine interoperability testing executed identical queries using Apache Spark, Apache Flink, Trino, and Presto to evaluate format portability and connector maturity across diverse execution engines. Instrumentation captured comprehensive metrics, including operation latency percentiles, resource utilization across CPU, memory, and network dimensions, metadata overhead ratios, and cost-relevant factors such as object storage API request counts that directly impact cloud infrastructure expenses.

The experimental design prioritized reproducibility and fair comparison by maintaining consistent cluster configurations, software versions, and workload characteristics across all format evaluations. Each test scenario executed multiple iterations with statistical analysis of results to account for measurement variance and ensure observed performance differences reflected genuine format characteristics rather than environmental noise. This rigorous methodology provides quantitative foundations for understanding architectural tradeoffs and operational implications of table format selection in production lakehouse deployments.

Scenario Category	Workload Description	Primary Metrics	Test Variations
Bulk Ingestion	Initial data loading from source files	Write throughput, metadata overhead, planning time	Parallelism levels range from low to high concurrency
Incremental Writes	Continuous micro-batch streaming updates	End-to-end latency percentiles, conflict rates	Batch sizes and arrival intervals
Concurrent Writers	Parallel writes to overlapping partitions	Conflict rate, retry behavior, and successful commits	Writer counts from minimal to extreme concurrency
Full Table Scans	Sequential reads of complete datasets	Read throughput, planning overhead, execution time	Query parallelism and caching strategies
Selective Queries	Predicate pushdown with filtering	Data scanned ratio, pruning efficiency	Single partition, range filters, complex predicates
Time Travel Queries	Historical snapshot access	Snapshot resolution time, metadata reads	Snapshot ages from recent to historical
Schema Evolution	Column additions, type changes, and partition evolution	Operation duration, backward compatibility	Different evolution types and impacts
Compaction Operations	Small file consolidation and optimization	Compaction throughput, file reduction, and impact	Target file sizes and bin-packing strategies

Table 2: Benchmark Scenarios and Evaluation Metrics [5, 6]

4. Results and Performance Analysis

Performance evaluation revealed significant differences across formats in bulk ingestion throughput and metadata management efficiency. Delta Lake demonstrated the highest write throughput during initial data loading, processing data substantially faster than Iceberg, while Hudi Copy-on-Write showed the lowest baseline performance. The performance gap widened considerably under high concurrency scenarios with one hundred concurrent writers, where Delta Lake maintained stable throughput while Iceberg experienced degradation attributed to manifest list contention during parallel metadata updates. Metadata overhead measurements confirmed Iceberg's efficiency in storage utilization, generating the smallest metadata footprint relative to total data size, while Hudi's embedded metadata approach resulted in proportionally higher overhead. Partition handling characteristics diverged notably for high-cardinality partitioned tables, where Iceberg's hidden partitioning architecture maintained consistent performance regardless of partition count, whereas Delta Lake and Hudi exhibited performance degradation as partition cardinality increased to thousands of leaf partitions.

Incremental write testing exposed distinct optimization strategies for streaming workloads and real-time data pipelines. Hudi Merge-on-Read tables achieved the lowest write latency for micro-batch ingestion by appending delta records to log files without rewriting base data files, followed by Iceberg with efficient manifest file updates, while Delta Lake showed higher latency variance attributed to periodic checkpoint consolidation every ten commits. Streaming consistency evaluation using Apache Flink demonstrated that all three formats successfully maintained exactly-once guarantees across extended test periods, though end-to-end latency from source to queryable data varied significantly. Hudi Merge-on-Read achieved the fastest source-to-query latency, while Delta Lake's Spark Structured Streaming integration exhibited approximately twenty percent higher latency due to micro-batch coordination overhead. Update and delete performance testing revealed Hudi's architectural advantages for modification-heavy workloads, processing substantially more updates per second compared to Iceberg and Delta Lake, though read amplification on Merge-on-Read tables without recent compaction penalized query performance by factors approaching three times baseline measurements.

Concurrent write handling revealed fundamental differences in transaction coordination strategies and scalability limits. At moderate concurrency with five parallel writers, all formats maintained minimal conflict rates below one percent with fast retry resolution. Increasing to ten concurrent writers exposed observable conflicts where Iceberg's optimistic concurrency control showed conflicts in the low single-digit percentage range with sub-second retry resolution, while Delta Lake exhibited slightly lower conflict rates but slower retry times attributed to transaction log validation overhead, and Hudi demonstrated the lowest conflict rate with longer resolution times reflecting its file-level conflict detection granularity. Beyond twenty concurrent writers,

all formats experienced performance degradation with distinct characteristics where Iceberg's manifest management became a bottleneck, showing exponential latency growth, Delta Lake's centralized transaction log created contention with more linear degradation, and Hudi maintained relatively stable throughput but required careful file sizing parameter tuning to avoid timeline service bottlenecks.

Read performance analysis demonstrated that full table scan throughput remained comparable across formats within single-digit percentage differences, all bounded by object storage bandwidth limitations rather than format-specific characteristics. However, query planning time showed substantial divergence, where Iceberg averaged the fastest planning for large tables through its hierarchical metadata structure, Delta Lake required additional time for transaction log scanning from the last checkpoint, and Hudi needed the longest planning duration due to timeline-based file discovery requiring iteration through commit history. Predicate pushdown efficiency testing revealed Iceberg's superior partition pruning capabilities, achieving the highest data reduction percentages through column-level statistics stored in manifest files, while Delta Lake achieved competitive but slightly lower reduction through partition pruning with file-level statistics, and Hudi demonstrated the least aggressive pruning due to coarser-grained statistics in timeline metadata.

Time travel query performance highlighted architectural implications where Iceberg's immutable metadata files enabled instant access to historical snapshots regardless of age with no observable planning time overhead, Delta Lake's incremental transaction log required scanning from last checkpoint to target timestamp adding latency that scaled with snapshot age, and Hudi's timeline compaction occasionally impacted historical reads when archival had occurred between snapshot creation and query execution. Cross-engine interoperability testing showed Iceberg achieving comparable performance across Spark, Trino, and Presto within ten percent variance reflecting its engine-agnostic design and mature connector implementations, Delta Lake queries in Trino showed thirty-five percent slower performance than Spark due to less optimized connector maturity, while Hudi support in Presto remained experimental with query planning taking multiple times longer than Spark equivalents and certain query types failing with compatibility errors.

Operation Type	Iceberg Performance	Delta Lake Performance	Hudi Performance	Key Observations
Bulk Ingestion Throughput	Moderate throughput, efficient metadata	The highest write throughput is maintained under concurrency	Lowest for CoW, competitive for MoR	Delta excels in initial loading scenarios
Metadata Overhead	Smallest footprint relative to data size	Moderate overhead from transaction log	Highest overhead with embedded metadata	Iceberg most storage-efficient
Query Planning Time	Fastest through the hierarchical structure	Moderate with log scanning overhead	Slowest due to timeline iteration	Planning efficiency impacts interactive queries
Partition Pruning Efficiency	Highest data reduction via column statistics	Competitive with partition-level pruning	Moderate with coarser statistics	Iceberg is superior for selective queries
Streaming Write Latency	Low latency with manifest updates	Higher variance from checkpointing	Lowest for MoR, highest for CoW	Hudi MoR is optimal for streaming ingestion
Update/Delete Throughput	Moderate performance	Moderate with file rewrites	Highest for MoR with log appends	Hudi excels in modification workloads
Concurrent Write Scalability	Exponential degradation beyond moderate concurrency	Linear degradation with log contention	Most stable with file-level conflicts	All formats struggle at extreme concurrency

Time Travel Access	Instant, regardless of snapshot age	Overhead scaling with snapshot age	Variable with compaction impacts	Iceberg immutable metadata is advantageous
Cross-Engine Compatibility	Comparable across Spark, Trino, Presto	Slower in non-Spark engines	Limited to experimental connectors	Iceberg is most portable across engines

Table 3: Performance Comparison Across Key Operations [7, 8]

5. Discussion and Architectural Tradeoffs

The observed performance characteristics stem directly from fundamental architectural decisions reflecting different design philosophies in distributed systems engineering. Iceberg's metadata-centric approach with its three-tier hierarchy optimizes for read performance and metadata evolution, enabling efficient pruning where only a small fraction of manifest files require evaluation for selective queries, proving optimal for read-heavy analytics workloads with moderate write concurrency and when cross-engine compatibility represents a critical requirement [8]. The immutability of Iceberg's metadata files provides strong consistency guarantees and efficient caching strategies, though manifest list updates require coordination that becomes contentious beyond moderate concurrency levels, suggesting this architecture performs best for workloads with high read-to-write ratios exceeding ten-to-one, where cross-engine portability justifies somewhat higher write coordination complexity.

Delta Lake's append-only transaction log architecture provides conceptual simplicity and strong consistency through a linearizable history of all table modifications, where every commit receives a unique sequence number establishing total ordering of operations [10]. This simplicity benefits operational teams by making table state reasoning straightforward and simplifying debugging of production issues, though checkpoint files that mitigate log scanning overhead introduce periodic performance variations during checkpoint creation, affecting workloads requiring consistent low-latency writes. The log's centralized nature creates a coordination point, becoming contentious under high write concurrency, explaining observed latency degradation at elevated concurrency levels, suggesting this approach suits organizations heavily invested in Spark ecosystems, prioritizing operational simplicity over extreme write scalability.

Hudi's dual table type architecture enables workload-specific optimization where Merge-on-Read tables excel for streaming ingestion, achieving substantially lower write latency by deferring compaction, while Copy-on-Write tables offer simpler operational characteristics at the cost of write amplification [9]. This flexibility suits organizations with diverse workload requirements, willing to manage additional operational complexity, including compaction scheduling and timeline maintenance. The file-level conflict detection reduces coordination overhead under high concurrency, explaining observed lower conflict rates, but at the cost of more complex conflict resolution requiring a deeper understanding of timeline service mechanisms. Hudi concepts emphasize incremental processing and upsert patterns where record-level indexes enable efficient point updates, making the format particularly well-suited for streaming architectures requiring sub-hour data freshness.

Format convergence trends reflect competitive dynamics where successful features propagate across implementations. Delta Lake introduced deletion vectors and liquid clustering, narrowing metadata efficiency gaps; Hudi adopted metadata tables providing hierarchical structures for improved query planning, and Iceberg implemented position deletes matching capabilities from competing formats. All three formats now support multi-table transactions through different implementation approaches, but provide comparable guarantees. Cross-engine ecosystem growth represents the most dramatic convergence, where Delta Lake's UniForm feature enables dual-format reads addressing Spark-centric limitations, major cloud providers announced multi-format support, reducing lock-in concerns, and connector maturity improved across Trino, Presto, and other engines for all formats.

Beyond raw performance metrics, operational considerations significantly impact production deployment success. Iceberg's separate metadata layer simplifies monitoring through clear operational metrics while its immutable files enable straightforward backup strategies. Delta Lake's human-readable JSON transaction log significantly simplifies debugging and troubleshooting production issues while its straightforward write path reduces training requirements. Hudi's multiple file types increase monitoring complexity but provide extensive timeline information useful for auditing and compliance requirements. Cost implications vary based on format characteristics, where Iceberg's efficient metadata and aggressive file pruning reduce storage costs through a smaller metadata footprint and more effective data skipping, while Hudi Merge-on-Read tables without aggressive compaction incur higher storage costs due to log file proliferation, and compaction requirements add compute costs for maintaining optimal performance characteristics.

Format	Optimal Use Cases	Key Strengths	Primary Limitations	Best Fit Organizations
Apache Iceberg	Read-heavy analytics, cross-engine workloads, high-cardinality partitions	Superior query planning, instant time travel, best partition pruning, engine-agnostic design	Manifest contention at high write concurrency, exponential degradation beyond 20 writers	Multi-engine environments, read-to-write ratios >10:1, cloud-agnostic strategies
Delta Lake	Spark-centric pipelines, operational simplicity, bulk ingestion	Highest write throughput, intuitive transaction log, operational simplicity, strong Spark integration	Checkpoint overhead, slower non-Spark connectors (35% penalty), linear degradation under concurrency	Spark-invested organizations, teams prioritizing simplicity over extreme scalability
Apache Hudi	Streaming upserts, modification-heavy workloads, sub-hour freshness	Lowest streaming latency (MoR), highest update throughput, workload-specific optimization	Operational complexity, read amplification without compaction (3x penalty), experimental engine support	Streaming architectures, CDC pipelines, and upsert-intensive applications requiring write flexibility

Table 4: Architectural Tradeoffs and Deployment Recommendations [9, 10]

Conclusion

This comprehensive empirical evaluation of Apache Iceberg, Delta Lake, and Apache Hudi at the terabyte scale reveals that table format selection involves nuanced tradeoffs rather than universally optimal choices across all dimensions. Iceberg emerged as the performance leader for read-heavy analytics workloads with superior query planning efficiency, exceptional partition pruning capabilities, and instant partition evolution through metadata-only operations. Its metadata-centric architecture and broad engine support, demonstrated through comparable performance across Spark, Trino, and Presto, make it optimal for organizations prioritizing cross-platform portability and read performance. However, manifest list coordination complexity under high write concurrency revealed scalability limitations requiring careful consideration for write-intensive workloads. Delta Lake demonstrated strengths in bulk ingestion throughput, operational simplicity through its intuitive transaction log design, and deep Spark ecosystem integration. Its straightforward operational model reduces cognitive load and training requirements while human-readable JSON transaction logs simplify production debugging and troubleshooting. The format's ongoing convergence toward engine-agnostic design through UniForm addresses primary vendor lock-in limitations, though at modest write performance costs. The popularity of Delta Lake shows that market forces have already proven the simplicity-first design philosophy to be correct, and it can be readily used by organizations that have interests in Spark ecosystems, where operational simplicity is more highly valued than extreme write scalability or multi-engine portability. Hudi distinguished itself through write flexibility via dual table types, enabling workload-specific optimization. Merge-on-Read tables achieved the lowest write latency for streaming workloads and the highest update throughput, validating its architectural approach for upsert-heavy scenarios requiring sub-hour data freshness. However, this flexibility requires managing additional operational complexity, including compaction scheduling, timeline maintenance, and careful parameter tuning. Read amplification on uncompacted Merge-on-Read tables represents a significant consideration requiring disciplined operational practices. The formats demonstrate clear convergence across core capabilities, including ACID transactions with snapshot isolation, time travel functionality, schema evolution support, and multi-table transaction guarantees. Competitive dynamics drive rapid feature adoption where performance gaps narrow over time, reducing format lock-in risks and enabling organizations to potentially switch formats or maintain multiple formats simultaneously. This convergence suggests practitioners should prioritize workload-specific characteristics and ecosystem constraints over seeking universally best formats. The quantitative evaluation supports selecting Iceberg for read-heavy analytics with cross-engine requirements, Delta Lake for Spark-centric operational simplicity, and Hudi for streaming upsert workloads requiring write-read tradeoff flexibility. As lakehouse architectures mature into dominant paradigms for data platforms, understanding format characteristics becomes essential for building scalable, cost-effective infrastructure. Cost implications revealed substantial compute savings for selective query workloads and storage efficiency gains through effective metadata management, demonstrating that format selection carries direct financial implications at scale.

Funding: This research received no external funding.
Conflicts of Interest: The authors declare no conflict of interest.
Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers.

References

[1] Databricks, "Data Lakehouse Architecture". [Online]. Available: https://www.databricks.com/product/data-lakehouse.

[2] Jeffrey Erickson, "What Is Apache Iceberg? Understanding Iceberg Tables," Oracle Corporation, 2025. [Online]. Available: https://www.oracle.com/autonomous-database/apache-iceberg/.

[3] Yongqiang He et al., "RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems," 2011 IEEE 27th International Conference on Data Engineering, 2011, https://ieeexplore.ieee.org/document/5767933.

[4] Anant Bhardwaj et al., "DataHub: Collaborative Data Science & Dataset Version Management at Scale," arxiv logo>cs>arXiv:1409.0798. 2014. [Online]. Available: https://arxiv.org/abs/1409.0798.

[5] Microsoft, "What is a data lakehouse?", 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/databricks/lakehouse/.

[6] Apache, "Apache HudiTM brings row-level updates/deletes to data lakes," [Online]. Available: https://hudi.apache.org/.

[7] University of California, Irvine, "AsterixDB," Asterix, [Online]. Available: https://asterix.ics.uci.edu/.

[8] Apache Iceberg, "Documentation," [Online]. Available: https://iceberg.apache.org/docs/latest/.

[9] Apache Hudi, "Concepts," [Online]. Available: https://hudi.apache.org/docs/concepts/.

[10] Databricks, "Delta Lake feature compatibility and protocols," 2024. [Online]. Available: https://docs.databricks.com/aws/en/delta/feature-compatibility.