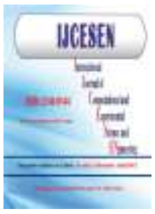
International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.1 (2026) pp. 572-581
http://www.ijcesen.com

ISSN: 2149-9144



Research Article

Modern Table Formats for Data Lakehouse Architectures: A Comprehensive Analysis of Apache Iceberg, Delta Lake, and Apache Hudi

Rahul Jain*

Cisco Systems Inc., USA
* **Corresponding Author Email:** rahul.j.profile@gmail.com - **ORCID:** 0000-0002-0047-4450

Article Info:

DOI: 10.22399/ijcesen.4867
Received : 02 December 2025
Revised : 30 January 2026
Accepted : 02 February 2026

Keywords

Lakehouse Architecture,
Acid Transactions,
Schema Evolution,
Streaming Ingestion,
Metadata Management

Abstract:

The transformation of enterprise data infrastructure has necessitated the creation of sophisticated table formats bridging the gap between traditional data lakes and data warehouses. Apache Iceberg, Delta Lake, and Apache Hudi have emerged as revolutionary technologies providing ACID transactional semantics, schema evolution, and advanced metadata management over cloud object storage systems. These formats address fundamental constraints of traditional data lake systems by delivering database-grade reliability without sacrificing cost-effectiveness and scalability of distributed storage. Each format embodies distinct architectural philosophies: Iceberg emphasizes engine neutrality with scalable metadata hierarchies, Delta Lake focuses on deep Apache Spark integration with optimized analytical query performance, and Hudi specializes in streaming ingestion patterns with efficient change data capture support. The architectural foundations include hierarchical metadata structures, transaction log mechanisms, and timeline-based state tracking, each presenting trade-offs in scalability, consistency, and operational complexity. Schema evolution capabilities enable structural adaptation without data rewrites, while sophisticated update and delete mechanisms using Copy-On-Write and Merge-On-Read strategies optimize for diverse workload characteristics. Streaming integration features facilitate real-time analytics through incremental query interfaces, native Kafka integration, and unified batch-streaming processing paradigms. However, these established formats encounter inherent overhead when handling true real-time workloads with millisecond-level latency requirements. Emerging technologies such as Apache Fluss and Apache Paimon represent next-generation solutions specifically architected for real-time data lake use cases, addressing limitations in existing frameworks through streaming-native architectures, unified streaming-batch storage engines, and optimized real-time query processing capabilities. Query optimization techniques, including hidden partitioning, data skipping, Z-order clustering, and comprehensive indexing subsystems, provide significant performance improvements for analytical workloads. The selection of appropriate table formats constitutes a foundational architectural decision with lasting implications for platform agility, operational complexity, and analytical capabilities, requiring careful evaluation of workload patterns, real-time requirements, ecosystem constraints, and strategic technology directions.

1. Introduction

The modern information landscape has witnessed a fundamental shift in how organizations architect their analytical infrastructure, moving away from inflexible, siloed systems toward more fluid and integrated models supporting diverse analytical requirements. Traditional data lakes, while providing cost-effective storage capabilities through columnar formats such as Parquet and ORC, have consistently failed to deliver the reliability and management features that contemporary enterprises demand. These conventional systems lack core database capabilities, including transactional consistency, schema flexibility, and effective metadata management, creating significant operational challenges for organizations striving to construct robust analytical platforms.

Apache Iceberg has emerged as a transformative solution, characterized as a game-changing table format specifically designed for big data analytics environments [1]. The format provides an open-source table specification introducing critical capabilities for managing large-scale data lakes, addressing numerous limitations inherent in traditional approaches. Iceberg's architecture fundamentally reimagines how metadata and data files interact, enabling organizations to achieve warehouse-like reliability on data lake storage. The lakehouse architecture concept represents a new generation of open platforms that fundamentally unify data warehousing and advanced analytics capabilities, eliminating traditional separation between these two paradigms [2]. This architectural evolution addresses the longstanding tension between systems optimized for business intelligence and those designed for machine learning and data science workflows.

Three dominant table formats have become central technologies in this lakehouse movement: Apache Iceberg, Delta Lake, and Apache Hudi. Each format introduces sophisticated metadata management layers built atop cloud object storage, enabling ACID transactional semantics, snapshot isolation, and time travel functionality. Despite fulfilling shared goals of achieving database-grade reliability in data lakes, these formats represent dramatically different architectural philosophies reflecting their origins and primary use cases. Apache Iceberg, initially developed at Netflix, emphasizes engine independence and scalable metadata architectures. Delta Lake, created by Databricks, focuses on profound integration with Apache Spark and efficient query execution for analytical workloads. Apache Hudi, originating from Uber's engineering teams, concentrates on streaming data ingestion patterns and effective change data capture handling.

While these established table formats have revolutionized data lake capabilities, they encounter inherent overhead when addressing true real-time workloads requiring sub-second latency. Batch-oriented architectures and periodic compaction cycles introduce delays incompatible with millisecond-level freshness requirements increasingly demanded by modern applications. Recognizing these limitations, the data engineering community has developed next-generation solutions specifically architected for real-time data lake scenarios. Apache Fluss represents a streaming-native storage system designed for real-time analytics with unified streaming and batch processing capabilities, eliminating architectural compromises present in batch-first designs. Apache Paimon introduces a streaming data lake platform combining high-throughput streaming writes with efficient batch query performance, bridging the gap between real-time ingestion and analytical processing through innovative storage engine design. These emerging technologies address fundamental constraints in established table formats, offering millisecond-level data freshness, optimized real-time query processing, and native streaming semantics without sacrificing analytical query performance.

This comprehensive examination analyzes the architectural foundations, operational characteristics, and performance profiles of established table formats alongside emerging real-time solutions, providing practitioners and researchers with a structured framework for evaluating their applicability to specific organizational requirements and workload patterns in modern data platform architectures.

2. Architectural Foundations and Metadata Management

The fundamental architecture of modern table formats determines their scalability boundaries, consistency guarantees, and operational complexity. Understanding these architectural choices proves essential for organizations selecting technologies that must support multi-petabyte datasets while maintaining query performance and data consistency.

Apache Iceberg introduces an innovative approach to metadata management through its hierarchical architecture, fundamentally separating concerns of tracking table state from the physical layout of data files [1]. This separation enables Iceberg to scale efficiently across massive datasets without encountering performance degradation typical of traditional catalog-based approaches. The format maintains a tree structure where snapshot metadata files reference manifest lists, which subsequently enumerate manifest files containing detailed information about individual data files. This multi-level indirection allows atomic commits through simple pointer updates at the snapshot level, avoiding expensive distributed coordination protocols. Iceberg's design philosophy emphasizes engine neutrality, ensuring diverse query engines can interact with tables without requiring format-specific adaptations. The metadata architecture supports efficient pruning during query planning, enabling engines to quickly identify relevant data files based on partition information and column-level statistics embedded within manifest structures. This capability becomes particularly critical in environments where tables contain millions of files distributed across thousands of partitions, scenarios increasingly common in modern data platforms.However, Iceberg's batch-oriented architecture introduces inherent overhead for real-time workloads. The snapshot-based commit model, while providing strong consistency guarantees, requires coordination overhead that limits transaction throughput for high-frequency updates. Compaction operations necessary to maintain optimal read performance introduce periodic processing delays, creating temporal gaps where newly written data remains inaccessible to readers. These architectural characteristics, while acceptable for minute-level freshness requirements, prove inadequate for millisecond-latency scenarios demanded by real-time applications.

The lakehouse architecture fundamentally transforms how organizations conceptualize data platform design by enabling direct analytical access to data stored in open formats on low-cost object storage [2]. This approach eliminates traditional requirements to maintain separate copies of data in warehouses and lakes, reducing both storage costs and operational complexity. The architecture achieves this unification through sophisticated metadata management, providing database-like capabilities including ACID transactions, schema enforcement, and efficient query planning over commodity storage systems. Modern table formats serve as the enabling technology layer, making lakehouse architectures practical, providing structured abstractions necessary for query engines to efficiently access data while maintaining consistency guarantees.

Delta Lake implements metadata management through a transaction log mechanism where each modification to a table appends an entry to an ordered log stored alongside data. This log-based approach provides straightforward audit trails of all changes and simplifies reasoning about table state evolution over time. The transaction log captures additions, removals, and metadata updates in JSON format, enabling both humans and systems to understand table history. To prevent unbounded growth of transaction logs, Delta Lake periodically generates checkpoint files consolidating cumulative effects of historical transactions into Parquet-encoded snapshots. These checkpoints enable query engines to restore existing table state without replaying full transaction history, maintaining tolerable query planning latency even for long-lived tables. The optimistic concurrency control protocol implemented atop this log structure enables multiple writers to safely commit changes concurrently, with conflict detection ensuring incompatible operations fail gracefully rather than corrupting table state.

Despite these sophisticated capabilities, Delta Lake's architecture introduces real-time processing overhead. The transaction log replay mechanism, while efficient for batch workloads, adds latency to read operations as log size grows between checkpoints. The OPTIMIZE and VACUUM operations required for maintaining performance necessitate exclusive table access periods, temporarily blocking writes and introducing unpredictable latency spikes. These architectural characteristics limit Delta Lake's suitability for applications requiring consistent sub-second latency guarantees.

Apache Hudi distinguishes itself through a timeline-based architecture explicitly modeling the temporal evolution of table state through sequences of discrete actions. The timeline tracks various operation types, including commits, compactions, cleans, and rollbacks, providing a comprehensive history of table modifications. Hudi supports two fundamentally different storage modes offering distinct trade-offs between write performance and read efficiency. Copy-On-Write mode generates new versions of modified files during updates, ensuring read queries never encounter merge overhead, but incurring significant write amplification. Merge-On-Read mode postpones merging operations by writing updates to separate delta logs, reducing write latency and storage overhead for update-intensive workloads while introducing merge complexity during query execution. This architectural flexibility enables organizations to optimize for specific latency and throughput requirements, though it introduces additional operational considerations around compaction timing and file management.

While Hudi advances real-time capabilities beyond traditional batch-oriented formats, it retains inherent limitations. The compaction processes required to maintain read performance in Merge-On-Read mode introduce background processing overhead, consuming cluster resources and occasionally blocking writes. The indexing mechanisms, while accelerating upsert operations, add memory overhead and update latency for maintaining index consistency. These architectural trade-offs, though acceptable for minute-level freshness, constrain Hudi's applicability to millisecond-latency scenarios.

3. Emerging Real-Time Data Lake Technologies: Apache Flussonet and Apache Paimon

Recognizing limitations of batch-first table formats for true real-time workloads, the data engineering community has developed next-generation technologies specifically architected for millisecond-level latency requirements. Apache Fluss and Apache Paimon represent innovative approaches to real-time data lake architectures, addressing fundamental constraints present in established formats through streaming-native designs and unified storage engines.

3.1 Apache Fluss: Streaming-Native Storage for Real-Time Analytics

Apache Fluss emerges as a streaming-native storage system designed specifically for real-time analytics workloads requiring sub-second data freshness. Unlike batch-oriented table formats retrofitted with streaming capabilities, Fluss adopts a streaming-first architecture where real-time processing represents the primary design consideration rather than an afterthought. The system provides unified streaming and batch processing capabilities through log-structured storage optimized for continuous data ingestion while maintaining efficient analytical query performance.

Fluss architecture centers on distributed log storage, enabling high-throughput append operations with minimal write latency. The system maintains separate log and table storage layers, where log storage optimizes for streaming writes with microsecond-level ingestion latency, while table storage provides columnar formats optimized for analytical queries. This separation enables Fluss to simultaneously satisfy contradictory requirements of streaming and batch workloads without architectural compromises inherent in unified designs. The log storage layer supports exact event-time ordering and deterministic replay semantics critical for complex event processing applications, capabilities often absent or inefficient in batch-first table formats.

Real-time query capabilities in Fluss leverage streaming-aware query planning that accounts for data freshness requirements during execution planning. The system maintains real-time indexes enabling millisecond-level point lookups and range scans over streaming data, eliminating query latency penalties associated with merging delta files in traditional Merge-On-Read architectures. Fluss implements continuous compaction processes that operate concurrently with read and write operations, avoiding blocking behaviors and latency spikes characteristic of periodic batch compaction in conventional table formats.

The streaming-native design philosophy extends to consistency guarantees, where Fluss provides exactly-once processing semantics without requiring external coordination systems. This contrasts with established table formats that often delegate streaming consistency to external frameworks like Apache Flink or Spark Structured Streaming. By internalizing consistency management, Fluss reduces operational complexity and eliminates coordination overhead that limits transaction throughput in batch-oriented designs.

3.2 Apache Paimon: Unified Streaming-Batch Data Lake Platform

Apache Paimon introduces a streaming data lake platform combining high-throughput streaming writes with efficient batch query performance through an innovative unified storage engine design. Paimon addresses fundamental limitations in existing table formats by treating streaming and batch as equal first-class workloads rather than optimizing primarily for one paradigm while accommodating the other as a secondary consideration.

The Paimon storage engine implements a multi-tier architecture where recent data resides in memory-optimized structures supporting millisecond-level writes and queries, while historical data transitions to columnar formats optimized for analytical processing. This tiered approach eliminates trade-offs between streaming write performance and analytical query efficiency that constrain conventional table formats. The system automatically manages data placement across tiers based on access patterns and freshness requirements, transparently optimizing for both real-time and historical query workloads without manual intervention.

Paimon's changelog mechanism provides native change data capture capabilities without the overhead associated with delta log merging in traditional Merge-On-Read implementations. The system maintains separate changelog streams tracking all data modifications, enabling downstream consumers to efficiently process incremental changes without scanning entire tables or merging complex delta structures. This architecture eliminates the read amplification characteristic of conventional MoR implementations, where query engines must merge multiple delta files with base data during scan operations.

The unified storage engine supports advanced features, including primary key constraints, aggregate materialized views, and lookup joins, capabilities typically requiring separate systems in traditional data lake architectures. Paimon implements these features through streaming-aware execution engines that incrementally maintain derived datasets as source data arrives, providing real-time materialized views without batch recomputation overhead. This native integration of streaming materialization contrasts sharply with established table formats requiring external processing frameworks to maintain derived datasets.

3.3 Comparative Analysis: Real-Time Capabilities

Apache Fluss and Apache Paimon address fundamental architectural limitations in established table formats regarding real-time processing. Traditional formats like Iceberg, Delta Lake, and Hudi, while providing streaming ingestion capabilities, retain batch-oriented architectures, introducing inherent latency overhead. Periodic compaction cycles, transaction log replay, and delta file merging create processing delays incompatible with millisecond-level freshness requirements.

Fluss and Paimon eliminate these overhead sources through streaming-native designs where real-time processing constitutes the primary architectural consideration. Continuous compaction, streaming-aware query planning, and log-structured storage enable these systems to provide millisecond-level data freshness without sacrificing analytical query performance. The unified storage engines avoid architectural compromises present in batch-first designs, supporting both real-time and historical workloads with equivalent efficiency.

However, these emerging technologies introduce trade-offs. The streaming-native architectures require more sophisticated operational management compared to established table formats. The multi-tier storage systems demand careful capacity planning and monitoring to ensure optimal data placement. The continuous compaction processes consume cluster resources that might otherwise serve query workloads. Organizations must carefully evaluate whether millisecond-level latency requirements justify the additional operational complexity compared to minute-level freshness achievable with established formats.

4. Transactional Semantics and Schema Evolution

Transactional capabilities over distributed object storage represent defining achievements of modern table formats, enabling reliable multi-user access patterns previously impossible in traditional data lake architectures. These guarantees prove essential for production analytical systems where data quality and consistency directly impact business decision-making.

Schema evolution represents the process through which data structures adapt over time to accommodate changing business requirements while maintaining compatibility with existing data and downstream consumers [3]. Modern analytical systems must gracefully handle schema modifications as organizations continuously improve their understanding of business entities, introduce new data sources, and increase analytical capabilities. Effective schema evolution strategies reduce disruption to running pipelines while ensuring historical data remains interpretable under existing schema definitions. This becomes more complicated in distributed environments with many consumers potentially accessing tables with various schema versions, necessitating careful coordination to ensure incompatibilities do not occur.

Apache Iceberg addresses these issues by implementing sophisticated schema versioning mechanisms as part of its metadata snapshot system. The format records and utilizes schema changes as explicit metadata modifications instead of making physical changes to underlying data, enabling schema changes to occur instantly even in petabyte-scale tables. Iceberg supports a wide variety of evolution operations, including adding and removing columns, rearranging them, and promoting and demoting types, with automatic validation ensuring proposed operations do not impair compatibility with existing data. The hidden partitioning mechanism further enhances flexibility by allowing partition specifications to evolve independently of schema changes, eliminating common sources of migration complexity in traditional systems.

Delta Lake provides robust schema enforcement and evolution capabilities specifically designed to prevent data quality issues while enabling controlled structural changes [4]. The format implements schema validation at write time, automatically rejecting data failing to conform to current table schemas unless explicit schema evolution options are enabled. This enforcement prevents silent data corruption scenarios where incompatible data might otherwise be written without validation, later causing query failures or incorrect analytical results. When schema evolution is explicitly requested, Delta Lake supports adding new columns with optional default values and widening column types to accommodate broader value ranges. The merge operation provides particularly powerful schema evolution capabilities, allowing upsert workloads to automatically expand target table schemas to accommodate new columns appearing in source data streams. Delta Lake's approach is flexible but safe, requiring opt-in for potentially disruptive schema changes while automatically implementing compatible modifications. The format maintains schema history in transaction logs, enabling time travel queries to accurately read historical data with schema versions in effect at query time, an essential feature for regulatory compliance and analytical reproducibility.

Apache Hudi applies schema evolution features to its ingestion pipeline, acknowledging that streaming data sources are often associated with schema drift, requiring real-time accommodation. The timeline-based architecture tracks schema versions alongside data commits, ensuring compaction operations correctly merge files created under different schema versions. Hudi supports backward-compatible schema changes through metadata propagation mechanisms, ensuring all table components remain synchronized as schemas evolve. The format's focus on incremental ingestion patterns requires especially robust schema handling since continuous pipelines cannot endure downtime associated with full table rewrites. Organizations using Hudi in change data capture situations enjoy the advantages of schema synchronization mechanisms that automatically identify upstream schema changes and propagate them downstream with minimal manual intervention.

For real-time workloads, schema evolution introduces additional complexity. The validation and propagation overhead associated with schema changes can temporarily impact write latency, particularly in high-throughput streaming scenarios. Emerging technologies like Apache Fluss and Paimon address these concerns through optimized schema validation pipelines and asynchronous propagation mechanisms that minimize impact on streaming write performance while maintaining consistency guarantees.

5. Update, Delete, and Compaction Strategies

The mechanisms through which table formats handle row-level modifications fundamentally determine their suitability for workloads involving frequent updates, a requirement increasingly common as organizations adopt near-real-time analytics and synchronize operational databases with analytical systems.

Apache Hudi was explicitly architected to address challenges of managing data lakes with frequent updates, use cases proving problematic for earlier table format implementations [5]. The format's dual storage mode architecture provides organizations with flexibility to optimize for either read performance or write efficiency based on workload characteristics. Copy-On-Write mode completely rewrites files containing modified records during each update operation, ensuring read queries encounter only fully consolidated data files without merge overhead. This method is optimal when working with read-intensive workloads where update rates are moderate, and query performance is more important than write latency. Merge-On-Read mode represents a radical approach to handling updates, where modifications are written to small delta log files instead of overwriting full base files, eliminating write operation amplification and storage costs by significant factors. This architecture enables sub-minute data freshness for streaming ingestion pipelines while maintaining acceptable query performance through efficient merge algorithms combining base files with delta logs during query execution. Hudi provides comprehensive automation for compaction processes necessary to maintain optimal file layouts in Merge-On-Read tables, with background services continuously monitoring file sizes and merge ratios to trigger compaction operations when thresholds are exceeded.

However, Hudi's compaction processes introduce real-time processing constraints. Background compaction consumes cluster resources, potentially impacting concurrent query and ingestion workloads. During compaction operations, affected partitions may experience temporarily elevated query latency as engines merge larger numbers of delta files. Organizations requiring consistent millisecond-level query latency must carefully tune compaction schedules and resource allocation to minimize impact on real-time workloads.

Deletion vectors represent storage optimization techniques that significantly accelerate delete operations by marking rows as logically deleted rather than physically removing them immediately [6]. This technique is especially useful in workloads with selective deletes affecting small fractions of files, typical in data retention processes and regulatory compliance procedures. Conventional delete schemes involve overwriting entire files to physically delete rows, costly operations causing large write amplification when only small fractions of rows require deletion. Deletion vectors eliminate this overhead by maintaining separate metadata structures tracking deleted row positions within each file, allowing query engines to filter deleted rows during scan operations without modifying base data files. Delta Lake has incorporated deletion vectors as optimizations for delete and merge operations, achieving order-of-magnitude performance improvements for operations affecting limited subsets of table data. The technique introduces trade-offs between write performance and read complexity, as query engines must merge deletion vectors with base files during execution, potentially impacting scan performance when deletion ratios become substantial. Organizations must balance these considerations by scheduling periodic optimization operations that physically apply deletion vectors and consolidate small files, maintenance patterns similar to database vacuum operations.

The accumulation of deletion vectors creates real-time query overhead as engines must evaluate additional metadata during scan operations. This overhead increases query latency proportionally to deletion vector count and complexity, constraining the format's suitability for latency-sensitive applications. Emerging technologies address these limitations through more efficient deletion tracking mechanisms that minimize scan-time overhead.

Apache Iceberg implements flexible deletion strategies through position deletes and equality deletes, providing multiple mechanisms for efficiently representing removed data. Position deletes specify exact row positions within data files that should be filtered during query execution, offering precise control for delete operations targeting specific records. Equality deletes define predicate-based filters, removing all rows matching specified column values, enabling efficient bulk deletion patterns. This dual approach accommodates diverse deletion patterns encountered in production workloads while maintaining acceptable query performance through optimized merge algorithms in supporting query engines. However, accumulation of deleted files gradually degrades both read performance and storage efficiency, necessitating periodic rewrite operations that physically remove deleted data and consolidate fragmented files into optimized layouts.

Similar to other established formats, Iceberg's delete handling introduces real-time processing overhead. The merge operations required during query execution add latency proportional to the delete file count and their size. Organizations with strict latency requirements must implement aggressive rewrite schedules to minimize delete file accumulation, increasing operational complexity and resource consumption.

6. Streaming Integration and Query Optimization

The convergence of batch and streaming processing paradigms represents recent trends in present-day data architecture due to business demands for increasingly fresh analytical information and real-time decision-making functionality. Table formats play central roles in enabling this convergence through native support for streaming ingestion patterns and incremental query interfaces.

Apache Hudi represents landmark achievements in bringing enterprise-scale data lake management capabilities to streaming-first architectures, as recognized through its graduation to top-level Apache project status [7]. The format emerged from Uber's operational requirements for ingesting massive streams of database change events into analytical data lakes with minute-level latency while maintaining full query consistency and efficient storage utilization. Hudi's incremental query capabilities enable downstream consumers to efficiently retrieve only records modified since specified commit timestamps, eliminating needs to repeatedly scan entire tables for change detection. This functionality proves essential for continuous ETL pipelines, materialized view maintenance workflows, and real-time feature engineering for machine learning systems. The extensive indexing subsystem comprising Bloom filters, hash-based indices, and B-tree structures allows record-level lookups to be performed much faster during upsert operations without scanning entire tables, which would make streaming ingestion impractical at scale. Hudi supports native integration with Apache Kafka for source-to-lake ingestion patterns, with automatic exactly-once semantics ensuring pipeline failures never produce duplicated or missing records.

While Hudi advances streaming capabilities significantly beyond traditional batch formats, inherent architectural constraints limit its applicability to true real-time scenarios. The minute-level latency achieved in production deployments, though impressive for data lake contexts, proves inadequate for applications requiring sub-second freshness. The periodic compaction processes necessary to maintain read performance introduce processing delays, creating temporal gaps between data ingestion and query availability. The indexing mechanisms, while accelerating upsert operations, add memory overhead and update latency for maintaining index consistency across distributed nodes.

Apache Iceberg has become an appealing option for organizations seeking to future-proof their data lake investments through open, engine-neutral designs and strong community support [8]. The wide support of the format across all query engines, including Apache Spark, Apache Flink, Trino, Presto, and Impala, gives organizations flexibility to change their technology preferences without undergoing costly migration efforts. The snapshot-based design of Iceberg automatically provides time travel ability, allowing analysts to query previous table states for reproducing analytics, regulatory compliance, and debugging. Partition evolution capabilities are distinguishing features of the format, meaning tables can change partitioning strategies without rewriting data, an important feature for long-lived analytical data whose access patterns change over time. Partitions can be rearranged as new dimensions are introduced or scaled to smaller time ranges based on shifting query patterns or switched to hourly partitioning based on changing patterns. The hidden partitioning abstraction eradicates frequent causes of query performance problems where users manually apply partition pruning predicates based on query predicates, ensuring user error does not result in costly full table scans.

However, Iceberg's batch-oriented architecture introduces real-time processing overhead. The snapshot commit mechanism, while providing strong consistency guarantees, requires coordination overhead, limiting transaction throughput for high-frequency updates. The metadata tree traversal necessary during query planning adds latency that, while negligible for batch workloads, becomes significant for latency-sensitive applications. Organizations requiring consistent sub-second query latency must evaluate whether Iceberg's architectural characteristics align with their real-time requirements.

Delta Lake is optimally scaled in Spark-focused settings via strong incorporation of Spark query planning frameworks and execution patterns of popular analytics designs. The data skipping properties of the format take advantage of file-level statistics to aggressively filter away irrelevant data at query planning, building selective queries with vastly smaller I/O requirements. Z-order clustering offers grouping of multi-dimensional data, sharing similar records across arrays of sort keys, which offers significant performance enhancements for queries with complicated multi-column filter predicates. These are especially useful in interactive analytical workloads where query latency directly influences user productivity and analytical iteration speed. The at-once support of Spark Structured Streaming helps Delta offer unified batch and streaming pipelines where both historical and current data are handled using the same code paths, allowing streamlined application development and management.

Despite these sophisticated optimizations, Delta Lake retains batch-oriented characteristics, introducing real-time processing constraints. The transaction log replay mechanism, while efficient for batch workloads, adds latency to read operations as log size grows between checkpoints. The OPTIMIZE and VACUUM operations required for maintaining performance necessitate exclusive table access periods, temporarily blocking writes and introducing unpredictable latency spikes. These architectural characteristics limit Delta Lake's suitability for applications requiring consistent sub-second latency guarantees.

6.1 Real-Time Query Optimization in Emerging Technologies

Apache Fluss and Apache Paimon address query optimization for real-time workloads through streaming-native architectures, eliminating overhead sources present in batch-oriented formats. Fluss implements streaming-aware query planning that accounts for data freshness requirements during execution planning, optimizing scan strategies based on whether queries target recent streaming data or historical analytical datasets. The real-time indexes maintained by Fluss enable millisecond-level point lookups and range scans over streaming data without the delta file merging overhead characteristic of traditional Merge-On-Read architectures.

Paimon's multi-tier storage architecture automatically optimizes query execution based on data location and access patterns. Recent data residing in memory-optimized structures supports millisecond-level queries without disk I/O overhead, while historical data in columnar formats benefits from traditional analytical optimizations, including predicate pushdown and columnar scanning. The unified query planning engine transparently spans both tiers, providing consistent performance regardless of data age or storage tier.

These emerging technologies demonstrate that true real-time query optimization requires architectural designs where streaming represents the primary consideration rather than a retrofit onto batch-oriented foundations. Organizations with strict latency requirements should carefully evaluate whether established table formats can meet their needs or whether streaming-native solutions like Fluss and Paimon offer more appropriate architectural foundations.

***Table 1**: Architectural Characteristics of Apache Iceberg and Lakehouse Platforms [1], [2]*

Characteristic	Apache Iceberg	Lakehouse Architecture
Metadata Organization	Hierarchical tree structure with snapshot files, manifest lists, and manifest files	Unified metadata layer providing database capabilities over object storage
Engine Compatibility	Engine-neutral design supporting Spark, Flink, Trino, Presto, Impala	Platform-agnostic, enabling multiple query engines on shared storage

Atomic Operations	Pointer-based atomic commits at the snapshot level	ACID transaction support eliminates data duplication requirements
Scalability Approach	Multi-level indirection avoiding distributed coordination	Direct analytical access to open formats on low-cost storage
Query Planning	Efficient pruning via partition information and column statistics	Schema enforcement and efficient planning over commodity systems

Table 2: *Schema Evolution Mechanisms in Modern Table Formats [3], [4]*

Feature	Schema Evolution Process	Delta Lake Implementation
Evolution Approach	Adapting data structures over time while maintaining compatibility	Schema validation at write time with explicit evolution controls
Backward Compatibility	Ensuring historical data remains interpretable under current schemas	Schema history is maintained in the transaction log for time travel
Modification Types	Column additions, deletions, reorderings, and type promotions	Adding columns with defaults, widening types, and merge-based expansion
Validation Strategy	Careful coordination to prevent incompatibilities across versions	Automatic rejection of non-conforming data unless explicitly enabled
Operational Impact	Minimizing disruption to pipelines during structural changes	Balancing flexibility with safety through opt-in mechanisms

Table 3: *Update and Delete Optimization Techniques [5], [6]*

Primary Strategy	Dual-mode architecture with CoW and MoR options	Logical deletion marking without physical removal
Write Optimization	MoR writes to the delta logs, reducing amplification	Separate metadata structures tracking deleted positions

Table 4: *Streaming Integration and Real-Time Capabilities [7], [8]*

Design Origin	Uber's operational requirements for minute-level latency	Engine-agnostic design with broad ecosystem adoption
Integration Patterns	Native Kafka integration with exactly-once semantics	Multi-engine support across Spark, Flink, Trino, Presto

7. Conclusions

The introduction of Apache Iceberg, Apache Delta Lake, and Apache Hudi as key technologies in the current data platform designs is one of the fundamental changes in the way organisations look at data analytics management. These tabular formats can effectively overcome the long-standing constraints of conventional data lake deployments by adding transactional assurances, schema flexibility, and advanced metadata control over cost-effective object stores. Although these formats have similar aims of introducing database-grade reliability to data lakes, the architectural philosophies produce significant distinctions in the characteristics of operation, performance, and workload appropriateness. Apache Iceberg also stands out with neutral principles of engine design and scalable metadata designs that support a wide variety of query engines and large-scale datasets, making it especially valuable to organizations with complex workloads on analytical processing that requires vendor-independence and long-term flexibility of architecture. Delta Lake brings unparalleled value to organizations that have made significant investments in Apache Spark environments, offering optimizations that are deeply integrated and operational patterns that are easy to operate with transaction log designs that offer strong ACID guarantees and significantly boost performance with data skipping and Z-order clustering of analytical workloads. Apache Hudi has been designed to address streaming ingestion and change data capture applications with a novel Merge-On-Read architecture and advanced indexing features, which reduce the write amplification of update-intensive workloads with automated maintenance functions that decrease the operational load of organizations with continuous ingestion pipelines that need sub-minute data freshness. The choice of proper table formats are foundational architectural choice with long-term ramifications, in terms of agility of platform, operational complexity, and analytics, based upon prudent considerations of workload patterns, ecosystem constraints, operational capabilities, and strategies in technology choices, not being universal choices. With the maturation of lakehouse architectures, further innovation grows the table format capabilities, enhances cross-engine interoperability, and offers more sophisticated analytical and machine learning workloads, making table formats a central enabling technology to the next-generation analytical platform, integrating batch processing, streaming ingestion, and advanced analytics into unified architectural structures to meet the needs of modern enterprise.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] BlueOrange Digital, "Apache Iceberg: A game-changer table format for big data analytics." [Online]. Available: https://blueorange.digital/blog/apache-iceberg-a-game-changer-table-format-for-big-data-analytics/

[2] Ali Ghodsi et al., "Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics," in Proc. 11th Biennial Conf. Innovative Data Systems Research (CIDR), 2021. [Online]. Available: https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf

[3] Dremio, "Schema evolution," 2024. [Online]. Available: https://www.dremio.com/wiki/schema-evolution/

[4] Databricks, "Diving into Delta Lake: Schema enforcement and evolution," 2019. [Online]. Available: https://www.databricks.com/blog/2019/09/24/diving-into-delta-lake-schema-enforcement-evolution.html

[5] Kuldeep Pal, "A beginner's guide to using Apache Hudi for data lake management," Walmart Global Tech Blog, Medium, 2023. [Online]. Available: https://medium.com/walmartglobaltech/a-beginners-guide-to-using-apache-hudi-for-data-lake-management-6af50ade43ad

[6] Databricks, "What are deletion vectors?" 2025. [Online]. Available: https://docs.databricks.com/aws/en/delta/deletion-vectors

[7] Uber Blog, "Apache Hudi graduation," 2020. [Online]. Available: https://www.uber.com/en-IN/blog/apache-hudi-graduation/

[8] Lindsay MacDonald, "Are Apache Iceberg tables right for your data lake? 6 reasons why," Monte Carlo Data Blog, 2024. [Online]. Available: https://www.montecarlodata.com/blog-are-apache-iceberg-tables-right-for-your-data-lake-6-reasons-why/

[9] Michael Armbrust, et al., "Delta Lake: high-performance ACID table storage over cloud object stores," ACM Digital Library, 2020. [Online]. Available: https://dl.acm.org/doi/10.14778/3415478.3415560

[10] Michael Armbrust et al., "Delta Lake: high-performance ACID table storage over cloud object stores," ACM Digital Library, 2020. [Online]. Available: https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf