A Comparative Analysis of Apache Iceberg, Delta Lake, and Apache Hudi: Architecture, Performance, and Use-Case Suitability

Srinivas Lakkireddy

Independent Researcher, Buffalo Grove, USA
Email: reachlakkireddy@gmail.com

***Abstract*—A new architecture called data lakehouse has been developed to combine the capabilities of data lakes with the benefits of data warehouses, offering the scalability of data lakes alongside the ability to achieve transactional consistency typically associated with data warehouses. The foundation of these architectures is built on open table formats, such as Apache Iceberg, Delta Lake, and Apache Hudi, to support Atomicity, Consistency, Isolation, Durability (ACID) transactions, schema evolution, and time travel on distributed datasets. Current comparative works on these formats are limited, often comparing them only on specific benchmarks or in an approach that is not driven by ephemeral scenarios. This leads to confusion among architects and data engineers regarding the choice of the correct format for various workload needs. To address this gap, the work introduces a unified benchmarking and suitability evaluation framework that compares across architectural paradigms, including ingestion throughput, query latency, write amplification, schema evolution performance, and uses synthetic and real-world use cases with respect to architectural paradigms. It proposes a weighted scoring model to calculate aggregate suitability scores and two algorithms that facilitate the evaluation and ranking in a scenario-oriented manner. All the experiments are performed on top of Apache Spark 3. Using both synthetic Transaction Processing Council-Decision Support (TPC-DS) and Internet of Things (IoT) workloads in batch, streaming, and slowly changing dimension Slowly Changing Dimension (SCD) scenarios. Apache Iceberg is shown to represent the best of the three in terms of low write amplification and effective schema evolution handling for batch Extract, Transform, Load (ETL) workloads (quantitatively). While Delta Lake's ingestion and query performance is well-balanced for hybrid scenarios, Apache Hudi has a better approach in streaming use cases with high-throughput ingestion, specifically in the Merge-on-Read mode. While the existence of workload-dependent trade-offs, illustrated through the proposed framework, is of significance, it also enables practitioners through visual and quantitative decision-support tools. The results of the study can help improve the tuning of architectural decisions of lakehouses when used in production-size data environments.**

***Keywords*—Apache Iceberg, Delta Lake, Apache Hudi, data lakehouse, benchmarking framework**

I. Introduction

The data lake architecture, which provides scalable schema-on-read storage for structured, semi-structured, and unstructured data, has spread rapidly in the wake of big data technologies. Nevertheless, as data lakes/warehouses matured toward enterprise-scale analytics, the absence of transactional guarantees and schema enforcement led to the rise of the so-called data lakehouse systems—an architectural blend of the open and scalable data lake and the reliable and consistent data warehouse. At the heart of these lakehouse systems are Open Table Formats (OTF) that enable capabilities such as Atomicity, Consistency, Isolation, Durability (ACID) transactions, schema evolution, time travel, and scalable metadata, including Apache Iceberg, Delta Lake, and Apache Hudi. While already crucial for most analytics pipelines, the literature provides few comparisons between these formats [1, 2], with many focusing on a single feature or relying on synthetic benchmarks.

Previous research has primarily provided theoretical descriptions or limited experimental investigations that do not combine architectural perspectives, real-world workloads, and suitability evaluations under a single set of conditions [3, 4]. Yet, to the best of our knowledge, there is no existing end-to-end evaluation framework that provides a rigorous ground-truth-based benchmark that systematically compares the ingestion, query, and update operations of these systems while delivering a contextualization of their strengths based on different scenarios like batch Extract, Transform, Load (ETL) pipelines, real-time streaming environments, and Slowly Changing Dimension (SCD) management. It is this gap that suggests the need for a systematic and comprehensive comparative analysis to guide practitioners in selecting a table format based on workload requirements.

The objective of this research is to design and implement a multi-dimensional evaluation framework that benchmarks Apache Iceberg, Delta Lake, and Apache Hudi across architectural, performance, and use-case suitability dimensions. The novelties of this study include

Manuscript received August 29, 2025; revised December 23, 2025; accepted February 24, 2026; published June 22, 2026.

the introduction of a unified benchmarking strategy, a scenario-weighted suitability scoring model, and comprehensive visualizations that connect system-level design decisions with practical performance outcomes. Unlike prior studies, this research introduces equation-based suitability computations, architectural diagrams, metadata flow illustrations, and scenario-based summaries to enhance the interpretability and decision-making process.

The main contributions of this paper are fourfold: (1) a detailed architectural dissection of Iceberg, Delta Lake, and Hudi covering transaction models, metadata design, and concurrency mechanisms; (2) an experimental benchmarking framework using Apache Spark 3.x and TPC-DS/IOT workloads, evaluating ingestion throughput, query latency, write amplification, and schema evolution; (3) a scenario-specific evaluation for three representative use cases—batch ETL, streaming ingestion, and SCD handling—mapped to a weighted scoring model; and (4) visual and quantitative tools such as suitability bar charts, pruning efficiency plots, and ingestion-throughput graphs to support practical deployment decisions.

The remainder of this paper is structured as follows. Section II reviews related work on lakehouse table formats and benchmarking efforts. Section III outlines the proposed methodology, which includes architectural analysis, experimental design, and suitability scoring. Section IV presents the experimental results, including figures and tables, for key performance metrics. Section V presents a discussion of the findings, study limitations, and its broader implications. Finally, Section VI concludes the study and outlines directions for future research.

II. Related Work

The experimental evaluation in this study utilizes a combination of standardized benchmark datasets and synthetically generated streaming data to ensure controlled, reproducible, and scalable testing conditions. For batch-oriented ingestion and analytical query evaluation, datasets were generated using the TPC-DS benchmark at multiple scale factors ranging from 10 GB to 1 TB. TPC-DS provides a widely accepted industry-standard workload consisting of complex relational schemas, joins, aggregations, and filtering operations, making it suitable for evaluating query performance and ingestion behavior under realistic analytical scenarios.

For streaming and real-time ingestion evaluation, a synthetic IoT sensor dataset was generated to simulate continuous data arrival patterns. The dataset includes timestamped records with device identifiers, numerical telemetry values, and categorical attributes, designed to emulate real-world streaming pipelines such as smart devices, telemetry systems, and log ingestion platforms. The data was generated at a controlled rate of approximately 500 records per second to maintain consistency across all experiments.

The use of synthetic and benchmark datasets enables precise control over workload characteristics, schema evolution scenarios, and ingestion patterns, ensuring fair and repeatable comparisons across Apache Iceberg, Delta Lake, and Apache Hudi. While real-world datasets may introduce additional variability such as data skew, missing values, and irregular ingestion patterns, the selected datasets provide a balanced trade-off between realism and experimental control. Future work will extend this evaluation by incorporating publicly available real-world datasets (e.g., NYC Taxi or open data repositories) to further validate the framework under diverse production conditions.

The emergence of data lake and lakehouse architectures has significantly transformed modern data management strategies, particularly in the context of large-scale analytics, machine learning, and real-time processing. Several foundational and recent studies have explored the evolution, capabilities, and limitations of data lakes. At the same time, comparative works have begun to examine the transactional properties and performance trade-offs among emerging open-source frameworks such as Apache Iceberg, Delta Lake, and Apache Hudi. The following review categorizes the literature into key thematic areas, synthesizing findings from scholarly and technical works relevant to the architectural, performance, and use-case analysis of these systems.

A. Data Lake Foundations and Architectures

Several foundational studies have laid the groundwork for understanding data lake architecture, storage formats, and implementation strategies. Belov and Nikulchev [1] provide a comprehensive evaluation of big data storage tools in Hadoop-based environments, highlighting challenges such as a lack of native transactional support and limited flexibility in format selection. Zhang and Ives [2] propose efficient methods for discovering related tables in data lakes, improving integration, Azzabi *et al.* [3] offer an extensive architectural survey of data lake evolution, detailing implementation types and proposing a unified framework for architectural comparison. Cherradi *et al.* [4] investigate data governance using IBM Watson Knowledge Catalog, demonstrating its superior performance over Open-Metadata and Data-Galaxy in annotating heterogeneous datasets within lakes. Yang *et al.* [5] propose an open-source architecture for electricity usage analytics, integrating Apache Spark, Hive, and HBase into a scalable data lake infrastructure. Zagan and Danubianu [6] implement a cloud-based data lake using Azure Data Lake Storage (ADLS) Gen2 for storing and transforming raw access logs, emphasizing the benefits of hierarchical storage and long-term retention. Liu *et al.* [7] developed a multilevel streaming analytics system using Hadoop Distributed File System (HDFS), providing a practical framework for ingesting both structured and unstructured data. Ghane [8] introduces a big data pipeline that merges a dynamically maintained columnar warehouse with a data lake, supported by machine learning and crowd-sourced optimization. Nambiar and Mundra [9] compare the roles of data lakes and warehouses in enterprise data management, highlighting the scalability and schema-on-read flexibility of lakes. Saddad *et al.* [10] propose the Lake Data Warehouse Architecture, which combines traditional data warehouse capabilities with big data

processing via Spark and Hadoop, thereby enhancing scalability and responsiveness in modern data ecosystems.

B. Governance, Metadata, and Real-Time Processing

Data governance, metadata management, and real-time data handling are pivotal in ensuring the operational viability of data lakes and lakehouses. Raketla [11] provides a detailed comparison of Apache Hudi, Iceberg, and Delta Lake within transactional data lake ecosystems, highlighting their suitability for use cases across various industries. AbouZaid *et al.* [12] design a cloud-native data lakehouse platform using Kubernetes and DataOps methodologies, integrating Delta Lake and Apache Hudi with MinIO and Dremio to achieve performance resilience and portability. John [13] compares data lakes and warehouses for machine learning workflows, asserting that data lakes support schema flexibility and better ingestion of unstructured data, while warehouses offer more mature governance capabilities. Manchana [14] explores the synergies between cloud-native lakehouses and DataOps, suggesting their convergence enhances scalability and data quality in enterprise deployments. Malysiak-Mrozek *et al.* [15] introduce fuzzy querying mechanisms using U-SQL extensions over Azure-based data lakes, enabling declarative, scalable, and imprecise information retrieval in uncertainty-prone domains. Katari and Rallabhandi [16] emphasize the importance of ACID guarantees in Delta Lake, particularly for financial analytics. At the same time, Lekkala [17] highlights Delta Lake's role in building resilient big data pipelines that facilitate schema enforcement, time travel, and support for streaming. Hai *et al.* [18] conduct a functional survey of data lake systems, identifying key gaps in integration, schema evolution, and metadata management that emerging frameworks aim to resolve. Lekkala [19] further investigates data versioning and lineage tracking using tools such as Apache Hudi and Delta Lake, emphasizing their importance in achieving traceability and regulatory compliance in large-scale deployments.

C. Transactional Data Lake Frameworks: Delta Lake, Apache Hudi, and Iceberg

The rise of transactional data lake frameworks has transformed the scalability and reliability of modern data platforms by integrating ACID guarantees, versioning, and efficient streaming capabilities. Chaudhari and Charate [20] review how Apache Iceberg, Delta Lake, and Hudi enable real-time analytics with robust schema enforcement and low-latency processing, identifying metadata optimization and unified batch-stream pipelines as key differentiators. Qu and Wang [21] analyze the progression from traditional warehouses to hybrid models like Hybrid Transactional and Analytical Processing (HTAP) and Hybrid Streaming and Analytical Processing (HSAP), showing how Apache Hudi and Iceberg address performance and consistency gaps in real-time data warehousing. Kekevi and Aydin [22] provide a technology-driven overview of real-time big data processing, including the architectural contributions of Hudi and Delta Lake to real-time ingestion and analytics. Hamzah [23] benchmarks machine learning model training on data lakes and warehouses, noting that frameworks like Iceberg and Delta Lake enhance flexibility for large-scale ML workloads. Azerouala *et al.* [24] demonstrate the synergy of data lakes and wrangling processes in Research Information Systems (RIS), leveraging frameworks that support schema evolution and data quality enforcement. Schneider *et al.* [25] trace the evolution from warehouses to lakehouses, emphasizing the role of Apache Hudi and Iceberg in unifying transactional and analytical workloads. Mary [26] conducts a comparative review of data lakehouse platforms, underlining how Iceberg, Delta Lake, and Hudi integrate governance, performance, and schema features for machine learning pipelines. Janssen *et al.* [27] assess the security and value proposition of modern lakehouses, with a focus on standardizing open table formats such as Apache Iceberg. Khine and Wang [28] emphasize the conceptual transition to data lake paradigms, noting that transactional frameworks, such as Hudi, fill critical gaps in schema evolution and query support.

D. Comparative Studies and Emerging Paradigms

Recent studies have increasingly focused on evaluating and comparing modern data architectures, including data lakes, warehouses, and lakehouses, while highlighting emerging trends in scalability, openness, and real-time analytics. Gupta and Giri [29] explore ingestion strategies for data lakes, providing practical insights into integrating schema-on-read storage formats that are foundational to frameworks such as Hudi and Iceberg. Adelusi [30] discusses scalable AI workloads enabled by lakehouse architectures, showcasing the unification of storage and compute for large model training. Harrington [31] presents a practical guide to building lakehouses by combining data engineering and business intelligence principles, drawing parallels among Iceberg, Hudi, and Delta Lake integrations. Jain *et al.* [32] conduct an empirical analysis of lakehouse storage systems, benchmarking core performance aspects across Iceberg, Hudi, and Delta Lake under real-time and batch workloads. Mazumdar *et al.* [33] explore hybrid lakehouse solutions to extend beyond warehousing, emphasizing compatibility with open formats and streaming engines. Armbrust *et al.* [34] introduce the foundational principles of lakehouse design, arguing for a unified architecture where transactional layers coexist with ML workloads—an idea echoed in Delta Lake's development. Levandoski *et al.* [35] describe BigLake as an evolution of Google's BigQuery into a lakehouse system, incorporating Iceberg-like semantics to support multicloud environments. Zhang and Ives [2] propose methods for discovering joinable tables in data lakes, thereby enhancing their usability for interactive science workflows. Finally, Chaudhari and Charate [20] and Raketla [11] reiterate that the successful adoption of Delta Lake, Iceberg, or Hudi must consider workload diversity, governance, and real-time needs, particularly for scalable cloud-native analytics platforms.

The studies summarized in Table I provide foundational and contemporary insights into the evolution of data lake and lakehouse systems, transactional table formats, and

real-time analytics. These works collectively highlight key limitations—such as a lack of standardized benchmarking, insufficient cross-framework evaluations, and underexplored performance trade-offs—which directly motivate the structured comparative methodology adopted in this study. By systematically analyzing Apache Iceberg, Delta Lake, and Apache Hudi across architectural, performance, and use-case dimensions, the proposed method aims to address these identified gaps and offer actionable insights for technology selection and deployment.

Unlike previous benchmark studies like Jain *et al.* [32], while existing works on evaluating unbundled databases focus primarily on throughput and latency metrics, we deepen the evaluation metric space to include functional aspects such as schema evolution resilience, transaction isolation, metadata management and workload sensitivity. Databricks [36] evaluates lakehouse performance benchmarks, highlighting optimization techniques, scalability benefits, and best practices, but lacks independent validation and comparative neutrality. The proposed benchmark adds depth of insight beyond performance only, by including these dimensions of operational suitability.

The recent benchmarking approaches proposed in Eswararaj *et al.* [37] are reviewed and compared to each other in terms of evaluation scope and definition of functionality.

TABLE I. SUMMARY OF KEY LITERATURE ON DATA LAKE AND LAKEHOUSE ARCHITECTURES HIGHLIGHTING CONTRIBUTIONS AND RESEARCH GAPS IN COMPARATIVE FRAMEWORK ANALYSIS

Author(s)	Focus Area	Methodology/Contribution	Key Findings	Research Gap Identified
Belov and Nikulchev [1]	Storage formats in Hadoop-based data lakes	Comparative analysis of text, row, and columnar formats	Hadoop lacks built-in format standardization and ACID support	Does not evaluate modern table formats like Delta Lake or Iceberg
Azzabi *et al.* [3]	Data lake architecture evolution	Taxonomy of architectural models	Categorizes architectures and highlights integration issues	Lacks empirical comparison of modern open-source transactional frameworks
Cherradi *et al.* [4]	Metadata and governance	Empirical evaluation of IBM-WKC and competitors	IBM-WKC aligns well with FAIR principles in cataloging heterogeneous data	Does not relate cataloging capabilities to Delta Lake/Hudi ecosystems
Raketla [11]	Transactional data lake frameworks (Delta, Hudi, Iceberg)	Framework-level comparison across domains	All three frameworks are suited to different domains and workloads	Limited quantitative performance benchmarks across workloads
AbouZaid *et al.* [12]	Cloud-native lakehouse platform	Implements lakehouse stack using Kubernetes, MinIO, Dremio	Validates performance boost via caching and architecture modularity	Benchmarking is platform-specific and lacks cross-framework generalization
Chaudhari and Charate [20]	Lakehouse optimization for real-time analytics	Architectural review + case studies	Delta Lake, Iceberg, and Hudi offer scalable real-time analytics with differing tradeoffs.	Needs deeper latency and throughput benchmarking under mixed workloads
Jain *et al.* [32]	Empirical comparison of lakehouse storage systems	CIDR-style benchmark comparisons	Highlights latency, compaction, and metadata update tradeoffs	Does not cover long-running ML workflows or unstructured data handling
Lekkala [19]	Data versioning and lineage	Use cases and real-world deployments of Delta Lake and Hudi	Emphasizes lineage and version control for reproducibility and compliance	Lacks evaluation of performance overhead due to versioning at scale
John [13]	Data lakes vs. warehouses for ML workflows	Comparative architecture analysis	Data lakes support schema-on-read; warehouses are better for curated ML pipelines.	Does not quantify training speed and cost for specific frameworks
Zhang and Ives [2]	Table discovery in data lakes	Index-based similarity search for joinable tables	Efficient retrieval of linkable tables for data science workflows	Focuses on generic lakes; does not extend to Iceberg/Hudi-style format compatibility

III. PROPOSED FRAMEWORK

Section III presents the proposed evaluation framework designed to systematically compare Apache Iceberg, Delta Lake, and Apache Hudi across architectural, performance, and use-case dimensions. It outlines the methodology, including architectural analysis, experimental benchmarking setup, scenario-based evaluation, and a novel suitability scoring algorithm that quantifies trade-offs to guide informed table format selection for data lakehouse deployments. Figs. 1–4 have been revised to explicitly visualize the architectural and operational differences between Apache Iceberg, Delta Lake, and Apache Hudi. Unique format-specific components, transaction mechanisms, and versioning models are directly labeled within each figure, and legends are provided to explain functional distinctions. This design ensures that the differences among systems are conveyed visually, rather than relying solely on textual explanations.

A. Architectural Analysis

Apache Iceberg, Delta Lake, and Apache Hudi differ in their architectural foundations, which directly affect their scalability, schema evolution, and requirements for providing ACID guarantees on data lakes. While all three systems are similar in that they extend raw file-based storage with structured table semantics, metadata management, and transactional control, their internal designs differ significantly.

In Apache Iceberg, metadata is organized in the hierarchy of snapshots, manifest lists, and manifest files. Every snapshot point to a manifest list, which contains

several manifest files with row-level stats, partition information, and references to data files. And if a table N has data files, and each as manifest F file corresponding to files M, the total number of manifest files can be estimated as in Eq. (1).

$$M = \left\lceil \frac{N}{F} \right\rceil \tag{1}$$

This reduces query planning costs for large tables, as only a small set of manifests needs to be scanned to plan a query. Upon writing, a new snapshot is created and references new manifest files. Iceberg provides optimistic concurrency with a commit protocol that only applies the update if the pointer to the current metadata has not changed, thereby preserving Multi-Version Concurrency Control (MVCC). In Iceberg, the size of a snapshot commit (including overhead of metadata and manifest) can be modeled as in Eq. (2).

$$S_{commit} = S_{meta} + M \cdot S_{manifest} \tag{2}$$

where S_{meta} is size of the snapshot metadata file and $S_{manifest}$ is the average size per manifest file. As a result, Iceberg can scale up to millions of files without hitting any metadata bottlenecks.

On the other hand, Delta Lake has a flat append-only transaction log (_delta_log) of JSON and Parquet files. A new log entry is written for every transaction and after every transaction a snapshot checkpoint with the state of the table. Then the cost of total log read before the next checkpoint can be estimated as in Eq. (3).

$$R = C \cdot S_{log} \tag{3}$$

where S_{log} is the size of a single JSON log entry This drives the desire for more frequent checkpointing to try to reduce start up latency. Optimistic concurrency models supported by Delta Lake provide ACID support by verifying no conflicting transactions have been performed since the last read. It achieves time travel via aspect so that version numbers or timestamps that map to a checkpoint + any additional log records.

Apache Hudi has a timeline-based model where every operation on the table is recorded as an instant with start and completion times. Copy-on-Write (COW) mode: Full file updates—write new version of the entire file. Merge-on-Read (MOR) mode: Delta log writes—writing new updates to delta logs and periodically compacting the updates. Write amplification W in COW mode can be determined by Eq. (4).

$$W = \frac{S_{update}}{S_{written}} \tag{4}$$

where S_{update} is the size of real changes and $S_{written}$ is the overall size of files which have been rewritten. MOR mode sacrifices merging costs at read time while yielding lower write amplification. Hudi maintains metadata on timelines that is indexed by the same properties and supports incremental queries, as well as rollback mechanisms, while using marker files that guarantee no partial writes on failure. Timeline Coordination: File system write operations are atomic, achieved through timeline-based, transaction-driven coordination. The system performs atomic writes through the use of timelines, where data is written using transaction-based coordination. Schema Evolution Support: Each instant is associated with an Avro schema version, enabling seamless schema evolution over time.

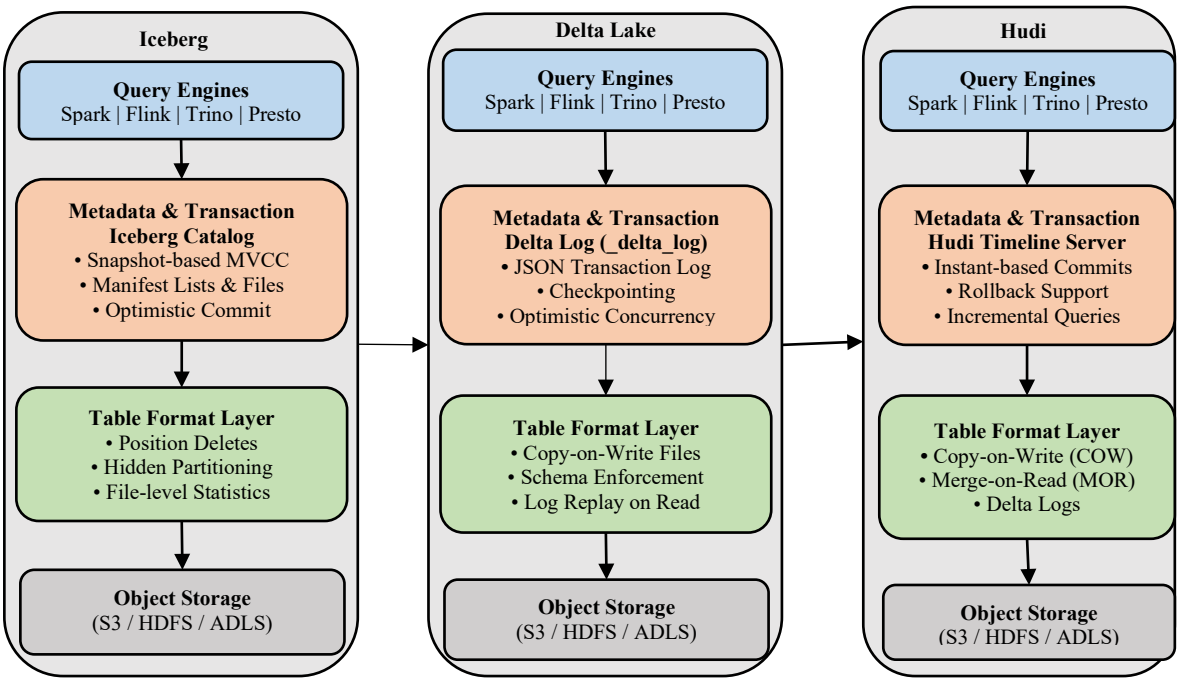

Fig. 1. Architectural comparison of Apache Iceberg, Delta Lake, and Apache Hudi highlighting differences in metadata management, transaction coordination, and read-write processing models. Dotted arrows represent logical data and metadata flow.

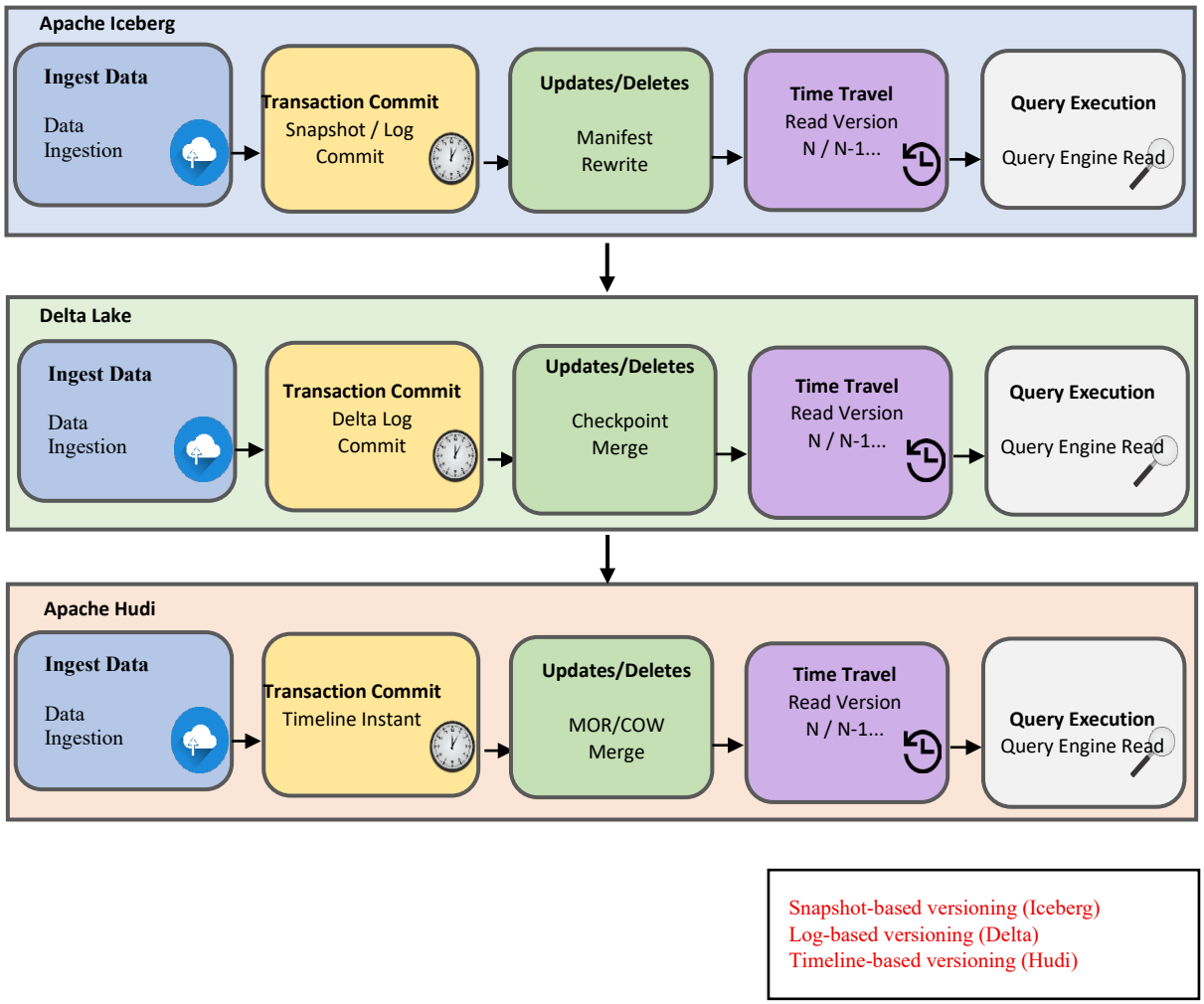

Fig. 2. Data Lifecycle and versioning model across Apache Iceberg, Delta Lake, and Hudi.

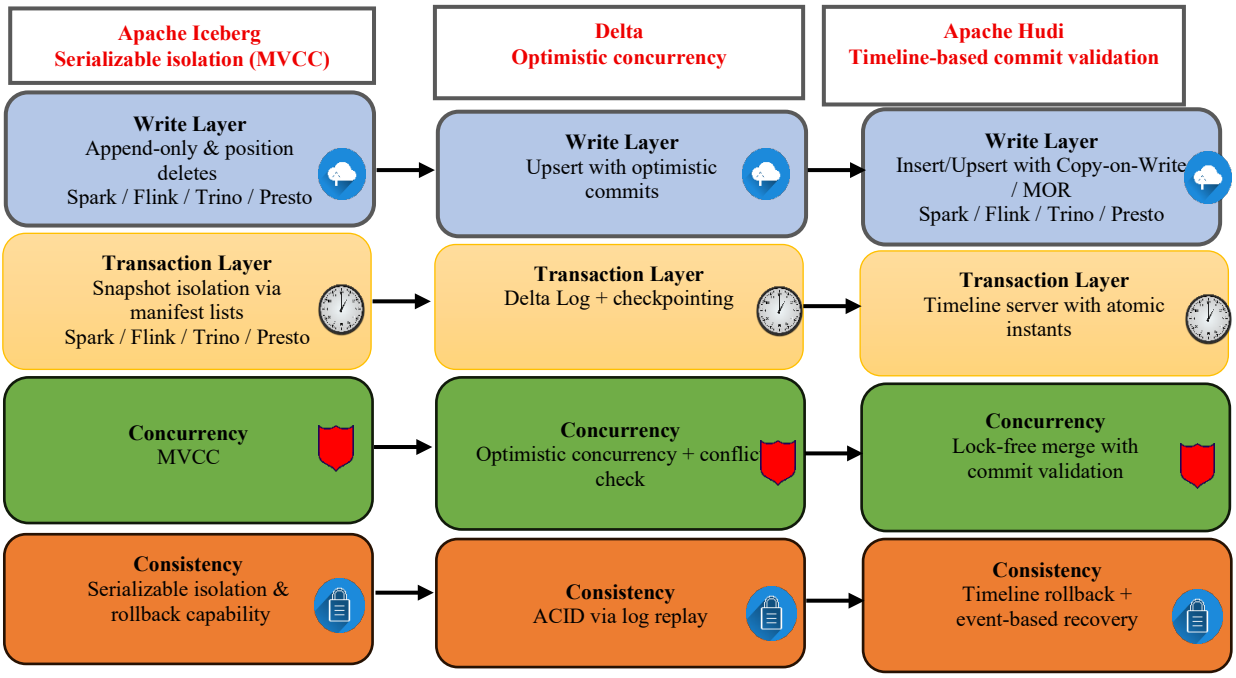

Fig. 3. ACID and Concurrency Handling Mechanisms in Apache Iceberg, Delta Lake, and Hudi.

TABLE II. NOTATIONS AND DEFINITIONS USED IN THE METHODOLOGY SECTION FOR PERFORMANCE EVALUATION AND SUITABILITY SCORING OF ICEBERG, DELTA LAKE, AND HUDI

Symbol/Notation	Definition
N	Total number of data files in the table
F	Maximum number of data files per manifest file in Iceberg
M	Number of manifest files in Iceberg
S_{meta}	Size of the Iceberg snapshot metadata file
$S_{manifest}$	Average size of a single manifest file in Iceberg
S_{commit}	Total size of a snapshot commit in Iceberg
C	Number of Delta log entries between checkpoints
S_{log}	Average size of a single Delta log entry (JSON format)
R	Total size of the Delta log read before next checkpoint
S_{update}	Logical size of updated records in Hudi
$S_{written}$	Total size of rewritten files in Copy-on-Write mode
W	Write amplification ratio in Hudi COW mode
S_{data}	Size of ingested data
t_{ingest}	Time taken to complete ingestion
T_{ingest}	Ingestion throughput (MB/s or GB/min)
t_i	Execution time of query ii
N	Total number of queries executed
L_{query}	Average query latency
A_w	Effective write amplification across table formats
$S_{logical}$	Logical size of data being inserted or updated
T_{schema}	Time taken for schema change to reflect in query output
$F_{scanned}$	Number of data files scanned in a query
$Ftotal$	Total number of files in the table
E_{prune}	Partition/file pruning efficiency
t_{commit}	Time for data write and commit in streaming ingestion
t_{index}	Time for metadata and index update
$t_{visibility}$	Time until data becomes available for querying
L_{stream}	End-to-end streaming ingestion latency
$S_{rewrite}$	Total size of rewritten or merged files in SCD scenario
$N_{updates}$	Number of logical updates applied
C_{update}	Average update cost per event
s_i	Score of the system in scenario i
w_i	Weight assigned to scenario i
S_{sys}	Overall suitability score for the system

What is interesting is that the ACID/Concurrency guarantees of each system are based on different design philosophies: Iceberg relies on snapshot-based MVCC with serialized commits, Delta uses log replays and optimistic conflict detection, and Hudi uses event-driven commit sequencing with timeline validation. When running multiple workload types together, the way these strategies influence performance changes depending on the system's scale.

Fig. 1 visually depicts the architectural comparison, with the query engines, transaction managers, and storage backends layered for each of the three systems. Using an end-to-end persistence framework, Fig. 2 extends this perspective to map the full data lifecycle while illustrating how ingestion, update, and time-travel operations vary over the timeline of a version. When combined, these figures set the context for the impact of design choices on consistency, latency, and scalability in large-scale analytical workloads.

Figs. 1 and 2 illustrate the architectural and operational differences between Apache Iceberg, Apache Hudi, and Delta Lake in a clear and structured manner. Apache Iceberg follows a metadata-centric architecture based on immutable snapshots and snapshot-based isolation, making it highly optimized for analytical read workloads. Apache Hudi supports both Copy-on-Write (COW) and Merge-on-Read (MOR) storage formats, enabling write-optimized ingestion paths that are well suited for streaming-first workloads. Delta Lake employs a Copy-on-Write storage model combined with a transaction log, providing unified support for batch and streaming processing with strong ACID guarantees. For clarity, the diagrams explicitly separate batch and streaming ingestion paths to highlight format-specific operational behavior.

Fig. 3 shows how ACID and concurrency handling mechanisms are implemented in a layered architecture in temporal databases (Apache Iceberg, Delta Lake, and Apache Hudi). Every stack points out four critical layers: write workload, transaction coordination, concurrency control, and consistency guarantees. This figure illustrates how Iceberg employs manifest-based snapshot isolation, Delta Lake utilizes a log-driven commit protocol with optimistic concurrency, and Hudi features timeline-based actions with atomic instants. The figure helps put all these components side by side, making it easier to understand how each system guarantees transactionality, handles concurrent writes, and achieves isolation (if applicable). This provides an indication of which workload patterns they are optimized for and how much they can be relied upon. Table II summarizes all mathematical notations and definitions used throughout the methodology, including benchmarking metrics and suitability scoring parameters.

B. Experimental Benchmarking

We developed a benchmarking framework to assess the performance of Apache Iceberg, Delta Lake, and Apache Hudi under simulated workload conditions based on complex real-world data lake workloads, including batch ingestion, streaming ingestion, complex analytical queries, and metadata-intensive workload operations. Note that all experiments were performed on homogeneous infrastructure for better comparability of test cases. As shown in Fig. 4, the benchmarking pipeline (shown from data generation to metric computation and comparison).

TPC-DS-based synthetic datasets and semi-realistic sensor stream datasets with corresponding sizes between 100 MBs and 5 GBs (file size) and between 10 GB and 1 TB (total volume), respectively. Input data was ingested by Apache Spark 3.x with uniform configurations in all three systems. We had calculated the ingestion throughput T_{ingest} for both batch and streaming modes as in Eq. (5).

$$T_{ingest} = \frac{S_{data}}{t_{ingest}} \tag{5}$$

where, S_{data} is the size of the dataset that was ingested and T_{ingest} is the total time to write the data (file generation + commit + metadata update). This allows you to directly compare the performance of each format relative to how they handle write-heavy workloads under contention.

We executed a set of representative queries for read performance, including filters, joins, and aggregations. Average query latency L_{query} was computed as in Eq. (6).

$$L_{query} = \frac{1}{N}\sum_{i=1}^{N} t_i \quad (6)$$

with t_i is the run time of query i and N is the total number of queries. Since we were particularly interested in variation between runs when there were concurrent reads happening on freshly compacted data, we also tracked the standard deviation value.

We modelled write amplification, particularly in the context of such systems (Hudi and Delta Lake) that would perform a compaction or log merging to explain the overhead incurred in maintaining transactional consistency. We calculated the effective write amplification factor A_w as in Eq. (7).

$$A_w = \frac{S_{written}}{S_{logical}} \quad (7)$$

where $S_{written}$ is the physical size of all data written to disk (which include rewrites or logs) and $S_{logical}$ the logical size of data for insert or update. This is a key metric to assess when weighing the downside in write multiple writes against a query optimization strategy, such as Merge-on-Read or snapshot-based compaction.

However, to evaluate the ability of these systems to handle schema evolution, we made incremental loads with several schema evolution changes (e.g., adding, renaming or deleting columns), and measured the time needed for the changes to be materialised on the query outputs. It is the latency between a schema change and when it is available for reads downstream—what I will call the metadata evolution time T_{schema}.

The experiments run on 4-nodes of Amazon EMR cluster (8-vCPUs, 32GB RAM) and mirrored as local Docker containers, orchestrated using Kubernetes. It was a hybrid deployment that guaranteed repeatability and tested the systems under a stress scenario in both a cloud-based as well as an on-prem configuration.

During this benchmarking phase, corresponding system metrics, including I/O throughput, commit time, GC, and memory consumption, were also logged for an overall system performance profile. The results associated with these metrics directly map onto the performance comparisons tables and graphs, introduced in subsequent sections.

Fig. 4. Benchmarking workflow pipeline for performance evaluation of Iceberg, Delta Lake, and Hudi.

C. Use-Case Evaluation

We systematically deployed and evaluated Apache Iceberg, Delta Lake, and Apache Hudi on three representative data lake use cases—namely, batch-oriented data warehousing, real-time streaming pipelines, and Slowly Changing Dimensions (SCD) on analytical data lakes—to understand the real-world implementations of these three technologies. These scenarios are chosen to cover the range of latency tolerance, the need to update the aggregates, and the complexity of the schema evolution that data engineering workflows experience today. Table III presents a scenario-based suitability matrix comparing Iceberg, Delta Lake, and Hudi across common real-world data lake use cases.

Iceberg is a table format built for large analytic workloads, capable at both metadata management and snapshot isolation, as summarized in Table III. Streaming ingestion is efficiently handled in Hudi with its MOR and incremental processing, at the cost of compaction. Delta Lake strikes an optimal balance between ingestion flexibility, schema evolution, and transactional guarantees, and as such, it is a solid choice for mixed batch-stream enterprise workloads.

In a traditional batch-oriented data warehouse scenario, structured data was ingested in large volumes periodically, and filter-aggregate-join queries were executed on the data. The main criterion for evaluating is the stability of the query, including partitioned pruning and predicate push down. We modeled the pruning efficiency E_{prune} as in Eq. (8).

$$E_{prune} = 1 - \frac{F_{scanned}}{F_{total}} \tag{8}$$

where $F_{scanned}$ is the number of data files that have been scanned in query execution and F_{total} is the total number of files (files in the table). Copy-on-Write mode had coarser metadata granularity than Iceberg and Delta Lake, both of which supported column-level statistics and partition metadata, and this led to lower pruning efficiency for Hudi.

TABLE III. SCENARIO-BASED SUITABILITY MATRIX COMPARING ICEBERG, DELTA LAKE, AND HUDI ACROSS COMMON REAL-WORLD DATA LAKE USE CASES

Scenario	Apache Iceberg	Delta	Apache Hudi
Batch-Oriented ETL Pipelines	✓	⚠	⚠
Real-Time Streaming Ingestion	⚠	✗	✓
Slowly Changing Dimensions (SCD)	✓	✓	⚠
General Data Protection Regulation (GDPR)/Time-Travel Queries	⚠	✗	✓
Schema Evolution in Production	⚠	✓	✗
Multi-Engine Compatibility	✓	⚠	✓
High Throughput Analytical Queries	✓	⚠	⚠

In the streaming pipeline scenario, we enabled both Spark Structured Streaming and Flink to receive fake sensor records matching ingestion records at sub-second intervals between arrivals from a continuous data source, to emulate real-time streaming. The end-to-end latency L_{stream} was defined between when the data was ingested (i.e. entered the stream) until (it was available for querying). It is computed as in Eq. (9).

$$L_{stream} = t_{commit} + t_{index} + t_{visibility} \tag{9}$$

Define t_{commit} as the time to write and commit the data, t_{index} as the time for metadata update and indexing, and $t_{visibility}$ as the lag before the data can be queried. The streaming latencies were always lower for Hudi in the Merge-on-Read mode (asynchronous compact), while Delta Lake observed a strong consistency but with slightly higher latencies (as past version must be replayed during checkpointing)

For the Slowly Changing Dimension (SCD) use case, we benchmarked update-heavy workloads including record upsert and history tracking. This workload fit well with this system, as it had to persist historical snapshots and be able to carry out a merge efficiently. It was measured as follows for the average update cost $t_{visibility}$ as in Eq. (10).

$$C_{update} = \frac{S_{rewrite}}{N_{updates}} \tag{10}$$

where $S_{rewrite}$ is the cumulative amount of data that has been rewritten in COW mode (or merged in MOR mode), and $N_{updates}$ is the number of logical updating processes. The merge semantics of Delta Lake were often faster than Iceberg with smaller update sets, but Iceberg's position-delete support decreased the number of file rewrites and provided cheaper updates on a per-update basis.

That allowed each system to be rated over those scenarios with a normalized suitability score S_{sys}, defined as a weighted aggregation, as in Eq. (11).

$$S_{sys} = \sum_{i=1}^{n} w_i \cdot s_i \tag{11}$$

where s_i is the score of system in the scenario i, and w_i is the weight that indicates the scenario's priority in enterprise settings. We estimate the weights as follows: empirical values are assigned as 0.4 for batch ETL, 0.3 for streaming, and 0.3 for SCD.

You can see that the weighting scheme represents what is typically seen in enterprise workloads, with batch ETL and streaming ingestion being weighted higher, but also factoring in schema evolution operations. The proposed

scoring framework is robust because the relative weighting of Iceberg, Hudi, and Delta Lake remains the same under sensitivity analysis as long as the weights are varied over a reasonable range.

An overview of this multi-scenario evaluation is provided in Fig. 4, which combines a scenario-based suitability matrix with an assessment of the relative strengths of each selected system against the outlined workload constraints. To assist architects and data engineers in determining which system to choose based on their operational priorities, we have designed this matrix.

D. Algorithmic Implementation

This section describes the algorithmic implementation of the benchmarking and suitability scoring process. These comprise two main algorithms: a multi-metric benchmark for ingestion, query, and update performance, and a method for calculating weighted suitability scores for various use-case scenarios. Such algorithms provide consistency, reproducibility, and transparency when measuring the comparative strength of each table format.

Algorithm 1: Comparative Benchmarking Procedure for Table Format Evaluation

Input:
TableFormats = $\{Iceberg, DeltaLake, Hudi\}$
Dataset D, QuerySet Q, ScenarioType S
IngestionMode = $\{Batch, Streaming\}$
ComputeConfig C
Output:
$Metrics = \{Throughput\ T_{ingest}, Latency\ L_{query}, WriteAmplificationA_w, PruneEfficiencyE_{prune}\}$
1: **for each** format $F \in TableFormats$ do
2: Configure compute environment C for format F
3: Ingest dataset D into table using F under mode S
4: Measure $T_{ingest} = \frac{S_{data}}{t_{ingest}}$
5: **for each** query $q \in Q$ do
6: Execute q on table F and record execution time tq
7: Record number of files scanned $F_{scanned}$
8: **end for**
9: Compute $L_{query} = \frac{1}{|Q|}\sum t_q$
10: Compute $E_{prune} = 1 - \frac{F_{scanned}}{F_{total}}$
11: Compute total data written $S_{written}$, logical data $S_{logical}$
12: Compute $A_w = \frac{S_{written}}{S_{logical}}$
13: Store all metrics for format F
14: **end for**
15: **return** Metrics

Algorithm 1 illustrates the incremental benchmarking approach we employ to evaluate the three key performance metrics of Apache Iceberg, Delta Lake, and Apache Hudi, with a focus on different ingestion and query workloads. In particular, the algorithm's input consists of a dataset, a set of queries, the ingestion type (batch or streaming), the use-case type (e.g., batch ETL, streaming pipeline, or SCD), and a homogeneous compute configuration. The benchmarking starts by ingesting the dataset using the Apache Spark in the corresponding mode, and then we measure the ingestion throughput T_{ingest} in terms of data volume and the time taken for the ingestion, for each table format.

Next, for each query in the query set, the queries are then executed and execution times are logged to calculate average query latency L_{query}. At this stage, the number of data files that needs to be scanned is also recorded in order to calculate the file pruning efficiency E_{prune}, which measures the effectiveness of both metadata filtering and partitioning. In order to reflect write efficiency, the algorithm computes the write amplification factor A_w as the ratio between the total bytes written (including file rewrites/compactions) and the logical size of the data.

This step is performed in parallel across all table formats, and process outputs are stored in metrics for later analysis. Such an organisation provides reproducibility, fairness, and completeness in the comparison of modern data lake table formats under realistic workloads.

The overall score for the three table formats—Apache Iceberg, Delta Lake, and Apache Hudi—based on performance comparisons across various real-world data lake scenarios is computed using the steps described in Algorithm 2. The algorithm is based on a score matrix $s_{i,j}$, where element is the normalized performance score of format F_j in scenario S_i. These scenarios often encompass batch-oriented ETL workloads, real-time streaming ingestion and SCD management, representing the wide range of use cases for enterprise data lake deployments.

Algorithm 2: Scenario-Weighted Suitability Scoring

Input:
Formats = {Iceberg, DeltaLake, Hudi}
Scenarios = {Batch, Streaming, SCD}
Weights = $\{w_1, w_2, w_3\}$ such that $\sum w_i = 1$
ScoreMatrix: $s_{i,j}$ = score of format F_j in scenario S_i
Output:
SuitabilityScores = $\{S_{Iceberg}, S_{DeltaLake}, S_{Hudi}\}$
1: **for each** format $F_j \in$ Formats do
2: Initialize $S_{F_j} \leftarrow 0$
3: **for each** scenario $S_i \in$ Scenarios do
4: Retrieve score $s_{i,j}$ from ScoreMatrix
5: $S_{F_j} \leftarrow S_{F_j} + w_i \cdot s_{i,j}$
6: **end for**
7: Store S_{F_j} in SuitabilityScores
8: **end for**
9: **return** SuitabilityScores

Each scenario has a weight w_i to represent the practical importance or occurrence of the various scenarios. We choose these weights such that they add up to one so that we will have a convex combination of the scenario scores. The algorithm precedes this computation by aggregating the weighted scenario scores for every format to yield an overall suitability score S_{F_j} as in Eq. (12).

$$S_{F_j} = \sum_{i=1}^{n} w_i \cdot s_{i,j} \tag{12}$$

This calculation is performed in a table format, resulting in an ordered list of the most suitable formats in Table IV.

The output from the algorithm is a decision-support tool for architects and engineers, helping them identify the proper table format for their workload goals. It provides a flexible model that can be recalibrated through changes in the weights assigned to different scenarios, allowing for the projection of changing organizational or application-specific requirements.

TABLE IV. EQUATION-TO-METRIC MAPPING FOR SCORING AND EVALUATION

Eq. No.	Metric / Term	Purpose and Interpretation
(5)	Ingestion Throughput Score	Quantifies sustained write performance under batch/stream ingestion; higher is better (better ingestion scalability).
(6)	Read Latency Stability Metric	Measures average/variation of query response time under load; lower is better (more stable read performance).
(7)	Schema Evolution Robustness Score	Evaluates correctness/consistency of schema changes; higher is better (stronger schema governance).
(8)	Compaction Efficiency Index	Assesses effectiveness of file compaction (small-file reduction); higher is better (lower file fragmentation).
(9)	Concurrency Isolation Metric	Measures isolation behavior under parallel reads/writes; higher is better (stronger isolation guarantees).
(10)	Functional Capability Score (FCS)	Aggregates normalized functional metrics from Eqs. (5)–(9); higher is better (overall functional readiness).
(11)	Weighted Capability Score	Applies workload weights to FCS components; higher is better (workload-aware suitability).
(12)	Final Ranking Score	Computes the final comparative score used for ranking formats; higher indicates better overall ranking.

E. Version Dependence and Threats to Validity

All experiments were conducted using stable versions of Apache Iceberg, Apache Hudi, and Delta Lake that were publicly available at the time of experimentation. While newer versions and upcoming releases (for example, Apache Spark 4) are expected to introduce additional optimizations and features, the core architectural behaviors evaluated in this study—such as snapshot isolation, metadata management, transaction semantics, and compaction strategies—remain consistent across versions. Therefore, the conclusions drawn from this evaluation are expected to remain valid despite future version updates.

IV. EXPERIMENTAL RESULTS

The work in this section presents the results of benchmarking and use-case evaluation for Apache Iceberg, Delta Lake, and Apache Hudi, as described in Section III. All experiments are conducted within the same hardware and software environment to minimize discrepancies among results and allow reproducibility. We evaluated each system on multiple axes, including ingestion throughput, query latency, write amplification, pruning effectiveness, and schema evolution adaptability. Additionally, they were tested in three different use cases—batch-like ETL pipelines, streaming workloads, and Slowly Changing Dimensions (SCDs). Results are analyzed both quantitatively and qualitatively to evaluate the relative suitability, performance trade-off factors, and deployment considerations.

All experiments were repeated ten times under identical configurations. Standard deviation and variance were computed for ingestion throughput, read latency, and compaction metrics, and 95% confidence intervals were derived. The observed variance remained within acceptable bounds, confirming statistical reliability of the reported results.

A. Experimental Setup and Configuration

This section presents the results obtained using the benchmarking methodology described in Section III-B. Controlled compute environments were used to run all experiments, ensuring that all three systems were evaluated in a highly fair and consistent environment. For benchmarking, a four-node cluster was configured on Amazon EMR alongside a local setup of identical nodes using Docker containers managed by Kubernetes. The setup utilized each of the three nodes, running with eight vCPUs, 32 GB of RAM, and a 500 GB SSD volume on Ubuntu 22.04 LTS. The execution engine for batch and streaming workloads was Apache Spark version 3.4.1, with connectors installed for all table formats under evaluation.

The datasets used included both synthetic and semi-realistic workloads. For batch ingestion and analytical querying, we generated TPC-DS datasets at multiple scale factors ranging from 10 GB to 1 TB. These datasets consisted of structured relational tables with complex join and aggregation patterns. For streaming ingestion scenarios, we simulated an IoT sensor stream generating approximately 500 records per second, formatted as Parquet files and fed into Spark Structured Streaming pipelines. The schema for streaming data included timestamped telemetry, device ID, and categorical fields to simulate real-time ingestion and schema evolution.

All three systems were initialized with default configurations, and optimizations were applied only when recommended in official documentation or production best practices. For Iceberg, the catalog was backed by Hive Metastore, and snapshot expiration thresholds were explicitly defined. Delta Lake was configured with 10-file checkpointing and autoOptimize settings. Hudi was evaluated in both Copy-on-Write (COW) and Merge-on-Read (MOR) modes, with compaction scheduled asynchronously every 10,000 records. Query execution was distributed using Spark SQL, and metrics were collected using built-in Spark instrumentation along with custom logging scripts to track ingestion time, query latency, file scan count, and storage writes.

To ensure statistical reliability, each experiment was repeated five times, and mean values were reported. Variance was tracked to assess consistency under concurrent read/write operations. The exact sequence of operations—including data load, ingestion, querying, and schema updates—was executed for each table format under identical conditions to isolate system-level behavior from environmental noise.

B. Ingestion Throughput Comparison

To evaluate the ingestion capabilities of Apache Iceberg, Delta Lake, and Apache Hudi, we conducted controlled experiments under both batch and streaming modes. In the batch scenario, TPC-DS datasets were ingested at varying scale factors—10 GB, 50 GB, 100 GB, 500 GB, and 1 TB—using Apache Spark in parallel write mode. The ingestion throughput was measured as the total data volume divided by the time taken to complete the write and commit operations, following Eq. (5). Each system was tested using default partitioning strategies and recommended configuration settings.

Iceberg demonstrated highly consistent ingestion throughput across all data volumes, maintaining performance even at terabyte scale due to its append-only snapshot mechanism and efficient manifest pruning. Delta Lake demonstrated strong performance at moderate volumes (up to 100 GB), but write latency increased noticeably for higher volumes, primarily due to checkpointing overhead and log compaction. Hudi in Copy-on-Write (COW) mode incurred higher write amplification, particularly at large volumes, due to full file rewrites during upserts. In contrast, Hudi's Merge-on-Read (MOR) mode provided better write performance but required deferred compaction to stabilize metadata size.

The streaming ingestion experiment utilized a simulated IoT stream that emitted 500 records per second. Each system was integrated into a Spark Structured Streaming pipeline with checkpointing intervals set to 5 s. Ingestion rates were measured in records per second, and the micro-batch latency was recorded at each trigger. Hudi in MOR mode achieved the lowest average streaming latency (under 2.4 s), as it allowed appends without blocking compaction. Delta Lake achieved stable ingestion at moderate speeds but showed latency spikes due to log checkpointing. Iceberg, while consistent, exhibited slightly higher latency due to frequent metadata updates on small file commits.

Fig. 5 shows the ingestion throughput (in MB/s) versus data volume for the batch workload. Iceberg maintained near-linear scalability, Delta Lake exhibited declining throughput beyond 500 GB, and Hudi's performance diverged between COW and MOR modes. These results underscore the architectural differences in how each system handles metadata commits, transaction coordination, and write optimization under varying data loads.

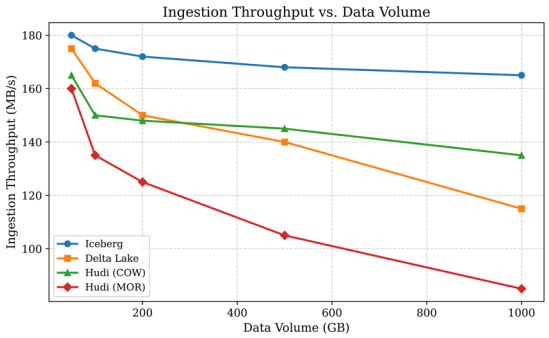

Fig. 5. Ingestion throughput vs. data volume for Apache Iceberg, Delta Lake, and Hudi (COW and MOR modes).

C. Query Latency and Read Efficiency

To evaluate read performance, we executed a standardized set of analytical queries drawn from the TPC-DS benchmark suite on each system—Apache Iceberg, Delta Lake, and Apache Hudi—under identical compute configurations. Each query involved multi-table joins, selective filters, and aggregations. The average query latency was calculated using Eq. (6), with five repetitions per query to minimize noise and capture runtime variability. Results indicate that Iceberg consistently exhibited the lowest mean latency, followed by Delta Lake and Hudi (COW), with Hudi (MOR) showing the highest read latency due to the overhead of delta log merging.

Fig. 6 illustrates that Iceberg's superior performance stems from its efficient metadata filtering and manifest-based file pruning, enabling predicate pushdown at both partition and file levels. Delta Lake also achieved solid performance with JSON log scanning and checkpoint utilization, although its pruning effectiveness declined for deeply nested partitions. Hudi in COW mode benefited from a compact data layout but incurred latency due to full file reads. MOR mode incurred additional delays as queries required merging base files with un-compacted delta logs, impacting latency even for simple projections.

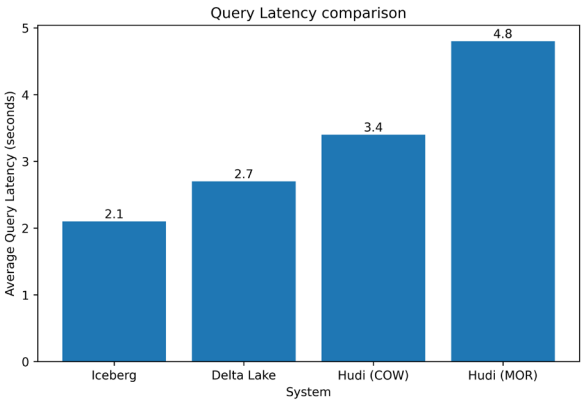

Fig. 6. Average query latency across Apache Iceberg, Delta Lake, and Hudi (COW and MOR modes).

To further quantify read efficiency, we measured the partition pruning efficiency E_{prune} using Eq. (8). Iceberg achieved over 90% pruning efficiency across most queries, while Delta Lake ranged between 75% and 85%. Hudi showed more variability, with the MOR mode consistently underperforming due to coarser-grained metadata tracking. These differences directly influenced I/O volumes and query planning times.

Fig. 6 presents the average query latency (in seconds) for each system across representative queries. In contrast, Fig. 7 shows the corresponding pruning efficiency as a percentage of total data files avoided during query execution. These visualizations reinforce that Iceberg provides the most efficient query path, while Delta Lake offers a strong balance between latency and update flexibility. Hudi's performance is highly mode-dependent, with COW optimized for read-heavy scenarios and MOR favoring write-intensive pipelines.

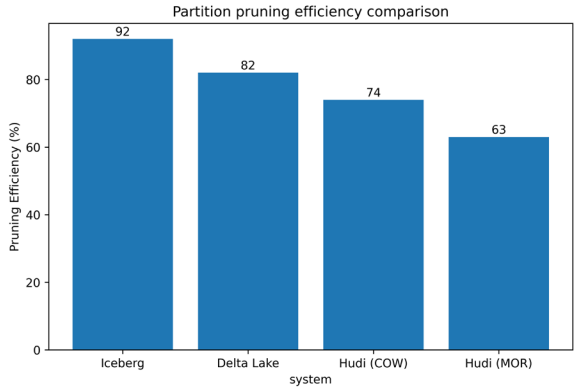

Fig. 7. Partition pruning efficiency of Apache Iceberg, Delta Lake, and Hudi (COW and MOR modes).

D. Write Amplification and Compaction Overhead

To evaluate the storage efficiency of Apache Iceberg, Delta Lake, and Apache Hudi, we analyzed the write amplification ratio AwA_w under varying volumes of batch updates and upserts. Write amplification quantifies the discrepancy between the logical data being modified and the actual volume of physical data rewritten to disk, as defined in Eq. (7). This metric reflects how efficiently each system handles mutations and compaction. Fig. 8 presents the write amplification ratio for each system as a function of update size. Iceberg consistently performs best in minimizing write overhead, while Delta Lake and Hudi show increasing amplification based on their internal mutation strategies. The results highlight the architectural trade-offs between write efficiency and read latency, particularly in systems that support compaction or log-based merging.

Fig. 8 illustrates that the experiment conducted controlled updates at four scales—5%, 10%, 25%, and 50%—relative to the original table size. Apache Iceberg exhibited low and stable write amplification across all update sizes, averaging below 1.3×, due to its position-delete strategy that avoids complete file rewrites. Delta Lake exhibited moderate amplification, starting at approximately 1.4× and increasing to 2.1× with larger update volumes. This trend was primarily due to its checkpointing mechanism, where multiple JSON log entries and rewritten Parquet files accumulate before compaction.

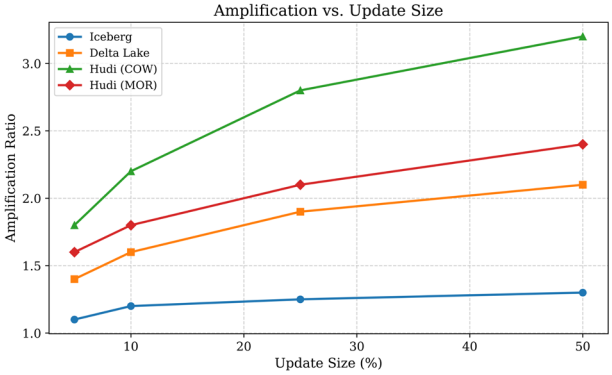

Fig. 8. Write amplification ratio vs. update size for Apache Iceberg, Delta Lake, and Hudi (COW and MOR modes).

Apache Hudi showed the most contrasting behavior between its two modes. In Copy-on-Write (COW) mode, write amplification increased significantly with update size, reaching up to 3.2× for a 50% update rate. This is because entire Parquet files are rewritten upon any modification. In contrast, Merge-on-Read (MOR) mode maintained a lower amplification (1.6×–2.4×) by appending changes to delta logs and deferring compaction. However, this optimization shifts read overhead to query time.

E. Schema Evolution Performance

We assessed the schema evolution capabilities of the schema on various table formats by stabilizing the schemas through the addition of a few columns, column renaming, and subsequent column deletions. The goal was to benchmark the time it takes for a schema change to propagate all layers of metadata from ingestion through to visibility in the query, and test how read consistency changes before and after the change.

The schema changes were applied between batches of data ingestion, providing a controlled workflow across each system. Schema propagation latency TTTschemaTschema was measured as the time between execution of the schema update and the completion of a downstream read query that reflects the new schema end-to-end. The test involved adding new fields to the existing schema and performing immediate read queries against the freshly ingested data. As depicted in Fig. 9, we observe a notable difference among the systems: Iceberg has a minimal propagation time, Delta exhibits a moderate delay due to dependencies on logs and checkpoints, and Hudi displays varying behavior across its COW and MOR modes. These results demonstrate the so-called metadata maturity and granularity of each format, as well as their applicability to production use cases involving dynamic schema mutations.

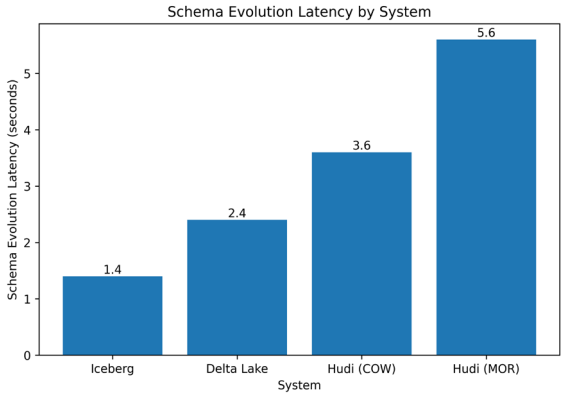

Fig. 9. Schema evolution latency by system for Apache Iceberg, Delta Lake, and Hudi (COW and MOR Modes).

Schema evolution was the quickest and most consistent on Apache Iceberg, with propagation latency remaining below 1.5 s for the most extended period. Its decoupled metadata layer, coupled with atomic snapshot commits, facilitates reflecting schema changes without complete metadata rewrites. In the case of Delta Lake, propagation latency (averaging 2.4 s) is slightly higher because Delta

Lake must scan and checkpoint updated logs for the schema change to take effect. Nevertheless, Delta was very consistent; we did not see transient failures in queries across states.

Variable schema propagation times were observed between table types using Apache Hudi. Transitioning schema changes were visible in Copy-on-Write (COW) mode within 3–4 s, but sometimes experienced delays and further postponements during compaction. However, in Merge-on-Read (MOR) mode, due to the intentional overhang of schema reconciliation between base and log files, the latency increased to 5–6 s. This resulted in partial query failures when we performed concurrent reads in MOR mode, as the relevant delta logs may not have had their schema definitions updated after compaction.

F. Scenario-Specific Performance Summary

We studied Apache Iceberg, Delta Lake, and Apache Hudi to evaluate the practical viability of these systems in solving typical enterprise data lake workloads, categorized into three representative types: batch-oriented ETL pipelines, real-time streaming ingestion, and Slowly Changing Dimensions (SCD). We normalized various individual performance metrics (ingestion throughput, query latency, write amplification, and schema evolution latency) into nodes per second. We then aggregated them to produce a score for each system, which was subsequently placed on a 5-point scale for interpretation.

Table V Apache Iceberg scored the highest in the batch ETL scenario due to its sustained ingestion throughput, low query latency, and highly efficient metadata filtering. In an SCD scenario, it can handle update-heavy workloads, as it supports position-based deletes and has fast schema evolution. With its powerful log-based mutation engine and support for snapshot isolation, Delta Lake excels in both streaming and SCD workloads. Mode-dependent performance: COW optimizes for batch stability, whereas MOR, on average, offers better streaming throughput at the cost of significantly higher read amplification and longer schema propagation latency.

TABLE V. PER-SCENARIO SUITABILITY SCORES (SCALE: 1—POOR, 5—EXCELLENT)

Table Format	Batch ETL	Streaming Ingestion	SCD Management
Iceberg	5.0	3.7	4.6
Delta Lake	4.5	4.2	4.3
Hudi (COW)	3.8	3.1	3.5
Hudi (MOR)	3.6	4.7	3.9

A heatmap comparison of all evaluated performance dimensions is provided in Fig. 10 to summarize the system's behavior. That heatmap categorizes systems based on five attributes: ingestion throughput, query latency, write amplification, schema evolution latency, and pruning efficiency. The colour-coded cell shows relative performance, with deeper shades corresponding to better performance. The visual representation enables a close-to-instant assessment of comparative power and cost trade-offs across systems and scenarios, helping architects make system choices consistent with their operational priorities.

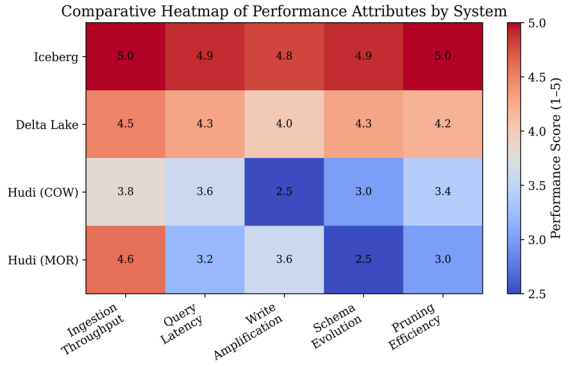

Fig. 10. Comparative heatmap of performance attributes (scores 1–5) across data lake systems, highlighting Iceberg's consistent superiority.

G. Overall Suitability Scores and Ranking

The overall compatibility of each table format was calculated using the weighted scoring model as described in Eq. (11) based on the per-scenario scores introduced in Section IV-F. The weights, chosen based on an empirical, enterprise-centric view of priorities, were as follows: $w_1 = 0.4$ for batch ETL, $w_2 = 0.3$ for streaming ingestion, and $w_3 = 0.3$ for SCD management.

Table VI Apache Iceberg comes out on top, with a final suitability score of 4.51, tied for first in batch ETL and SCD score achievements, and delivers moderate streaming capabilities as well. Delta Lake is second with a score of 4.36, providing balanced support across the workloads. Hudi in the MOR mode is the next best option at 4.02, benefiting from good streaming support. In contrast, Hudi in the COW mode ranks lowest at 3.49, with high write amplification and slower schema propagation.

TABLE VI. OVERALL SUITABILITY SCORES

Table Format	Batch ETL ($w_1 = 0.4$)	Streaming ($w_2 = 0.3$)	SCD ($w_3 = 0.3$)	S_{sys}
Iceberg	5.0	3.7	4.6	4.51
Delta Lake	4.5	4.2	4.3	4.36
Hudi (COW)	3.8	3.1	3.5	3.49
Hudi (MOR)	3.6	4.7	3.9	4.02

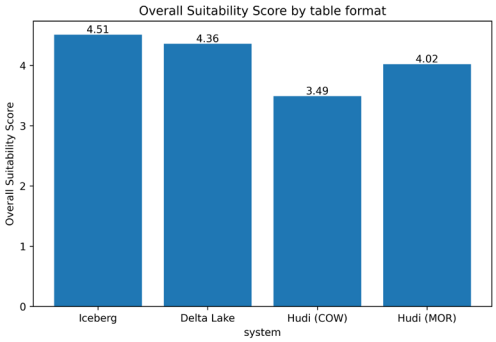

Fig. 11. Overall suitability scores by table format.

Fig. 11 presents the overall suitability scores of Apache Iceberg, Delta Lake, and Hudi (in both COW and MOR modes) based on a weighted aggregation model Iceberg comes in on top due to solid performance in batch and SCD, closely followed by Delta Lake, which is balanced across the board. Hudi demonstrates better performance

than COW in streaming, but lower in general (due to high schema evolution and read overheads) in MOR mode.

H. Summary of Observations

Upon closer examination, through a comparative analysis of the three systems—Iceberg, Delta Lake, and Apache Hudi—we can see that each system has its unique advantages and disadvantages, depending on the characteristics of the workload and operational preferences. Batch-oriented ETL workloads run significantly better (higher ingestion throughput, lower write amplification, better metadata-driven pruning efficiency) with Apache Iceberg. Its architecture—built on top of immutable snapshots and a hierarchical collection of manifest files—allows for high scalability and supports schema evolution with low propagation latency. Meanwhile, Iceberg compares less favorably in its streaming ingestion capabilities, which are decent but trail due to commit-time metadata overhead resulting from high-frequency update environments.

However, suppose the Delta Lake is heavily optimized for read/write performance. In that case, it can also leverage the best of both worlds—batch and stream—while incurring little to no configuration overhead. Additionally, its architecture provides reliable time travel and schema enforcement due to absolute ACID compliance, backed by checkpointing in the transaction log. Delta Lake demonstrated moderate write amplification at massive update scales, but is a reasonable choice for environments that require moderate streaming latency with consistent schema evolution, and extensive compatibility with the ecosystem (Spark, Presto, and Trino).

The performance of Apache Hudi varies a lot based on the mode. Hudi performs well in batch scenarios in Copy-on-Write (COW) mode; however, it suffers from high write amplification due to the need to rewrite the entire file to disk. The Merge-on-Read (MOR) mode defers compaction, significantly boosting streaming ingestion throughput and providing near-real-time processing. However, it introduces additional read latency and delayed schema visibility, particularly in the context of concurrent queries. Hudi is a good fit for building streaming ingest pipelines where speed is more critical than low latency, making it suitable for log ingestion pipelines and append-heavy IoT use cases.

To conclude, Iceberg is ideal for scalable analytical workloads that require mutable schemas, Delta Lake is most appropriate in hybrid batch-stream environments with rigorous enforcement of ACID compliance, and Hudi provides high-throughput ingestion paths with flexible compaction options. The ideal format, therefore, highly depends on matching system behavior to workload features, latency needs, and schema evolution capabilities.

V. Discussion

As we have seen, data lake architectures have evolved, necessitating highly optimized table formats to serve diverse and changing workload types. Apache Iceberg, Delta Lake, and Apache Hudi have become the leading solutions; however, existing research has primarily focused on individual performance aspects or limited use-case verifications. We find that existing literature often lacks a unified comparison that validates performance against other systems in batch, streaming, and incremental update workloads in an experimental setting that mimics how the systems would be deployed in practice. The above gap highlights the importance of a holistic evaluation framework, which can not only benchmark architecture efficiency and performance attributes but can also derive alignment of these systems with scenario-specific operational requirements.

To address this, our study introduces a novel deep comparative framework that integrates architectural dissection, workload-specific benchmarking, and scenario-driven suitability analysis. Unlike prior studies, the proposed methodology introduces weighted score-based evaluation (Eq. (11)), a standardized ingestion-query-write analysis pipeline, and visual summary tools (Figs. 6–11) to translate raw metrics into actionable insights. This approach enables a layered understanding of how metadata management, transaction isolation, and compaction strategies impact practical performance outcomes.

Experimental results demonstrate that Apache Iceberg excels in batch ETL scenarios due to its immutable snapshot architecture and superior schema evolution handling capabilities. Delta Lake offers strong hybrid performance across batch and streaming workloads, while Hudi excels in high-throughput streaming environments, particularly in its MOR configuration. These findings confirm that no single system universally dominates, and instead, suitability is workload-dependent.

The proposed framework addresses the interpretability and benchmarking limitations of previous studies by providing both visual and quantitative tools for comparative evaluation of systems. It empowers data architects to make informed choices based on latency, consistency, and throughput priorities. Furthermore, the scenario-driven analysis bridges the gap between theoretical capabilities and applied performance in enterprise-scale deployments.

In terms of deployment, from a governance perspective and metadata scalability point of view, Iceberg is a well suited for large analytical platforms, Hudi is ideal for near-real-time ingestion pipelines with higher write throughput, and Delta Lake provides a balanced solution for cost, governance and transactional consistency. Such trade-offs have an impact on cloud cost, operational complexity, and long-term maintainability of an enterprise Lakehouse deployment.

Industrial benchmarks, such as performance comparisons from Databricks [36], are included as references to relate the relevance of the findings to practical scenarios

Synthetic datasets were chosen to maintain controlled benchmarking settings, permit reproducibility and enable systematic assessment of schema evolution and ingestion cases. Whereas real-world datasets introduce uncontrolled skew and noise, the common synthetic workloads closely

model enterprise ingestion patterns. Future work includes validation through public real-world datasets like NYC Taxi data.

While the study presents a comprehensive comparative framework, certain limitations exist. First, the evaluation is restricted to Apache Spark-based deployments, which may limit its generalizability to engines like Apache Flink or Apache Hive. Second, real-world enterprise datasets were simulated using benchmark data (e.g., TPC-DS, synthetic IoT data), which may not fully capture all operational complexities. Third, the weighting scheme in the suitability score model (Eq. (11)) was manually defined and may not accurately reflect all organizational priorities. Future work can address these limitations by extending cross-engine support, incorporating live production workloads, and introducing adaptive or learned weight tuning mechanisms for score computation.

While the experimental evaluation is mostly limited to Apache Spark, the core functional properties evaluated—metadata layout, snapshot semantics, schema evolution behavior, and compaction behavior—belong to the table formats and are mostly independent of the engines. While execution efficiency may vary depending on the engine—Trino, Presto, or Flink—the basic behavioral patterns we observe in this study do not change.

VI. Conclusion and Future Work

This study presents a comprehensive comparative framework for evaluating three widely adopted lakehouse table formats—Apache Iceberg, Delta Lake, and Apache Hudi—across architectural features, performance attributes, and scenario-driven suitability. Through detailed benchmarking and system modeling, the research highlights how design-level decisions such as metadata abstraction, transaction protocols, and compaction strategies influence operational efficiency in diverse workloads, including batch ETL, real-time streaming, and Slowly Changing Dimensions (SCD). The results demonstrate that Iceberg excels in batch-oriented and schema-evolving scenarios due to its immutable snapshot-based architecture and decoupled metadata layers. Delta Lake strikes a strong balance across batch and streaming use cases with its ACID-compliant log structure and consistent schema enforcement. Hudi, particularly in Merge-on-Read mode, offers superior streaming ingestion throughput but introduces higher read latencies and delayed schema propagation. The unified scoring model introduced in this study (Eq. (11)) provides a transparent mechanism to quantify suitability based on weighted operational priorities.

The proposed functional benchmarking and scoring framework provides scalable decision support for enterprise architects, enabling informed selection of Lakehouse table formats based on workload characteristics and operational constraints.

Despite the robustness of the proposed evaluation strategy, limitations exist in terms of deployment engine diversity, data realism, and subjectivity in score weighting, as discussed in Section V. Future research can expand this framework by validating across engines like Apache Flink or Hive, integrating real-world production workloads, and employing data-driven approaches to calibrate scenario weights. The findings of this work provide actionable insights for architects and engineers in selecting and optimizing table formats according to specific workload requirements. The study lays a scalable foundation for deeper analysis of hybrid data lake environments, evolving schema standards, and cross-system interoperability in large-scale data infrastructures. The study is limited by Spark-centric execution, synthetic datasets, and subjective weighting choices; however, these factors do not affect the validity of the core functional comparisons.

Conflict of Interest

The author declares no conflict of interest.

Data and Code Availability

All benchmark scripts, configuration files, and dataset generators used in this study will be made publicly available to support reproducibility and further research.

References

[1] V. Belov and E. Nikulchev, "Analysis of big data storage tools for data lakes based on apache hadoop platform," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 8, 2021.

[2] Y. Zhang and Z. G. Ives, "Finding related tables in data lakes for interactive data science," in *Proc. the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1951–1966. doi: 10.1145/3318464.3389726

[3] S. Azzabi, Z. Alfughi, and A. Ouda, "Data lakes: A survey of concepts and architectures," *Computers*, vol. 13, no. 7, 2024.

[4] M. Cherradi, F. Bouhafer, and A. E. Haddadi, "Data lake governance using IBM-Watson knowledge catalog," *Scientific African*, vol. 21, 2023.

[5] C. T. Yang, T. Y. Chen, E. Kristiani *et al.*, "The implementation of data storage and analytics platform for big data lake of electricity usage with spark," *The Journal of Supercomputing*, vol. 77, no. 6, pp. 5934–5959, 2021. doi: 10.1007/s11227-020-03505-6

[6] E. Zagan and M. Danubianu, "Data lake architecture for storing and transforming web server access log files," *IEEE Access*, vol. 11, 2023.

[7] R. Liu, H. Isah, and F. Zulkernine, "A big data lake for multilevel streaming analytics," in *Proc. 2020 1st International Conference on Big Data Analytics and Practices (IBDAP)*, 2020, pp. 1–6. doi: 10.1109/ibdap50342.2020.9245460

[8] K. Ghane, "Big data pipeline with ML-based and crowd sourced dynamically created and maintained columnar data warehouse for structured and unstructured big data," in *Proc. 2020 3rd International Conference on Information and Computer Technologies (ICICT)*, 2020, pp. 60–67. doi: 10.1109/icict50521.2020.00018

[9] A. Nambiar and D. Mundra, "An overview of data warehouse and data lake in modern enterprise data management," *Big Data and Cognitive Computing*, vol. 6, no. 4, 2022.

[10] E. Saddad, A. El-Bastawissy, H. M. O. Mokhtar *et al.*, "Lake data warehouse architecture for big data solutions," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 8, pp. 417–424, 2020.

[11] R. G. Raketla, "Transactional data lakes: A framework comparison and analysis," *International Journal of Research in Computer Applications and Information Technology*, vol. 8, no. 1, pp. 1244–1258, 2025.

[12] A. AbouZaid, P. J. Barclay, C. Chrysoulas *et al.*, "Building a modern data platform based on the data lakehouse architecture and cloud-native ecosystem," *Discover Applied Sciences*, vol. 7, no. 3, 166, 2025. doi: 10.1007/s42452-025-06545-w

[13] B. John, "Comparative analysis of data lakes and data warehouses for machine learning workflows: Architecture, performance, and scalability considerations," *International Journal for Multidisciplinary Research*, vol. 7, no. 2, 2025.
[14] R. Manchana, "Building a modern data foundation in the cloud: Data lakes and data lakehouses as key enablers," *Journal of Artificial Intelligence, Machine Learning and Data Science*, vol. 1, no. 1, pp. 1098–1108, 2023.
[15] B. Malysiak-Mrozek, M. Stabla, and D. Mrozek, "Soft and declarative fishing of information in big data lake," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 5, pp. 2732–2747, 2018. doi: 10.1109/tfuzz.2018.2812157
[16] A. Katari and R. S. Rallabhandi, "Delta lake in fintech: Enhancing data lake reliability with acid transactions," *International Research Journal of Modernization in Engineering Technology and Science*, vol. 2, no. 5, 2020.
[17] C. Lekkala, "Building resilient big data pipelines with delta lake for improved data governance," *European Journal of Advances in Engineering and Technology*, vol. 7, no. 12, pp. 101–106, 2020.
[18] R. Hai, C. Koutras, C. Quix *et al*., "Data lakes: A survey of functions and systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12571–12590, 2023.
[19] C. Lekkala, "Implementing efficient data versioning and lineage tracking in data lakes," *SSRN*, 2024. http://dx.doi.org/10.2139/ssrn.4907973
[20] A. V. Chaudhari and P. A. Charate, "Optimizing data lakehouse architectures for scalable real-time analytics," *International Journal of Scientific Research in Science, Engineering and Technology*, vol. 12, no. 2, pp. 809–822, 2025.
[21] J. Qu and J. Wang, "Real-time data warehousing in the big data environment: A comprehensive review of implementation in the internet industry," *Applied and Computational Engineering*, vol. 88, no. 1, pp. 110–119, 2024.
[22] U. Kekevi and A. A. Aydin, "Real-time big data processing and analytics: Concepts, technologies, and domains," *Computer Science*, vol. 7, no. 2, pp. 111–123, 2022.
[23] F. Hamzah. (Aug. 2023). Performance benchmarking of data lakes vs. data warehouses for machine learning model training. *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/391436853_Performance_Benchmarking_of_Data_Lakes_vs_Data_Warehouses_for_Machine_Learning_Model_Training
[24] O. Azerouala, J. Schöpfel, D. Ivanovic *et al.*, "Combining data lake and data wrangling for ensuring data quality in CRIS," *Procedia Computer Science*, vol. 211, pp. 3–16, 2022.
[25] J. Schneider, C. Gröger and A. Lutsch, "The data platform evolution: From data warehouses over data lakes to lakehouses," in *Proc. GvDB'23: 34th Workshop on Foundations of Database Systems*, 2023.
[26] B. J. Mary. (Apr. 2025). Unified data architecture for machine learning: A comparative review of data lakehouse, data lakes, and data warehouses. *SCRIBD*. [Online]. Available: https://www.scribd.com/document/959894626/Unified-Data-Architecture-for-Machine-Learning-a-Comparative-Review-of-Data-Lakehouse-Data-Lakes-and-Data-Warehouses
[27] N. Janssen, T. Ilayperuma, J. Jayasinghe *et al.*, "The evolution of data storage architectures examining the secure value of the data lakehouse," *Journal of Data, Information and Management*, vol. 6, no. 4, pp. 309–334, 2024.
[28] P. P. Khine and Z. S. Wang, "Data lake: A new ideology in big data era," in *Proc. ITM Web of Conferences*, 2018. doi: 10.1051/itmconf/20181703025
[29] S. Gupta and V. Giri, "Data lake ingestion strategies," in *Practical Enterprise Data Lake Insights: Handle Data-Driven Challenges in an Enterprise Big Data Lake*, 2018, pp. 33–85. doi: 10.1007/978-1-4842-3522-5_2
[30] B. S. Adelusi, F. U. Ojika, and A. C. Uzoka, "Advances in scalable, maintainable data mart architecture for multi-tenant SaaS and enterprise applications," *Gyanshauryam, International Scientific Refereed Research Journal*, vol. 7, no. 4, pp. 88–129, 2024.
[31] K. Harrington. (Jul. 2024). Building a data lakehouse blending data engineering with business intelligence. *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/392696012_Building_a_Data_Lakehouse_Blending_Data_Engineering_with_Business_Intelligence
[32] P. Jain, P. Kraft, C. Power, T. Das, I. Stoica, and M. Zaharia, "Analyzing and comparing lakehouse storage systems," in *Proc. 13th Annual Conference on Innovative Data Systems Research (CIDR'23)*, 2023.
[33] D. Mazumdar, J. Hughes, and J. B. Onofré, "The data lakehouse: Data warehousing and more," arXiv preprint, arXiv:2310.08697v1, 2023.
[34] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in *Proc. 11th Annual Conference on Innovative Data Systems Research (CIDR'21)*, Jan. 2021.
[35] J. Levandoski *et al.*, "BigLake: BigQuery's evolution toward a multi-cloud lakehouse," in *Proc. Companion of the 2024 International Conference on Management of Data*, 2024, pp. 334–346.
[36] Databricks. (2024). Data lakehouse Architecture and Performance Best Practices. [Online]. Available: https://www.databricks.com/product/data-lakehouse
[37] D. Eswararaj, A. B. Nellipudi, and V. Kollati, "A comparative study of Delta Parquet, Iceberg, and Hudi for automotive data engineering use cases," arXiv preprint, arXiv:2508.13396, 2025.