AMERICAN ACADEMIC PUBLISHER

INTERNATIONAL JOURNAL OF DATA SCIENCE AND MACHINE LEARNING (ISSN: 2692-5141)

Volume 05, Issue 02, 2025, pages 126-144
Published Date: - 03-10-2025
DOI - https://doi.org/10.55640/ijdsml-05-02-12

Evaluating Effectiveness of Delta Lake Over Parquet in Python Pipeline

Sai Nikhil Donthi
Department of Software Engineering, University of Houston Clear Lak .
Oil and Gas Industry Houston Texas, USA

ABSTRACT

We have been witnessing rapid growth of data-intensive applications adopting efficient columnar storage formats, with Parquet becoming a widely used standard in modern data pipeline. Parquet has been efficient than traditional databases in the aspect of columnar storage, evolution of schema, efficient compression, and vast range of supported tools. While Parquet still had issues of capturing transactional logs and didn't support ACID properties or atomic writes, which resulted in data corruption and metadata operations became expensive. Delta Lake, an open-source storage layer built on Parquet, addresses these limitations by introducing ACID transactions, schema evolution, time travel, and unified batch and streaming support. The study conducted on evaluating effectiveness of Delta Lake over Apache Parquet helps to gather and take advantage of the key benefits that delta lake provides in the world of big data and the optimization techniques used in Microsoft fabric as the baseline with delta lake being the backend storage bin. With the extensive increase of data driven critical applications in the IT sector, the columnar storage orientated data formats such as Apache parquet have become the industry standard in the big data world. Though parquet can compress and store different data formats like Json, xml, audio, csv, etc but it lacks the ACID properties that delta lake offers. Parquet formats offer high data compression efficiency and rapid query execution on the other hand lacks schema enforcement, reliability, and transaction guarantees which is crucial in this modern world. In short Delta Lake, an extension of parquet is an open-source storage layer introduced on top of parquet with delta logs that store incremental transaction logs helps to eliminate the limitations of parquet by adding ACID transactions, schema evolution, time travel and unified batch streaming support. This research inquires that parquet offsets to read low optimal workloads (does not support parallel operations and require batch processing) while delta lake provides remarkable advantages for heavy workloads that require data versioning, reliability, and parallel execution. This research draws the effectiveness of Delta Lake in comparison to Parquet by reviewing critical performance parameters such as read/write speed, concurrency management, update/delete efficiency, and operational reliability, within both batch and streaming data (parallel) processing in a Python-based data pipeline.

KEYWORDS

Delta Lake, Parquet, ACID Transactions, Schema Evolution, Transaction Log, Time Travel, Unified Batch and Streaming Support, Optimistic Concurrency Control, Data Versioning, Metadata Management, Update/Delete Performance, Big Data Reliability, Streaming Data Processing, Data Lakehouse Architecture, Parallel Read/Write Operations, Data Compression Optimization.

I. INTRODUCTION

In the recent years, exponential growth and evolution of data applications have changed the landscape of data engineering and analytics. When organizations started processing historical or real-time data it became difficult to use traditional database which used row to row storage method, traditional relational database and SQL databases performed well with minimal number of columns and fixed processing of schema. Less storage cost and faster I/O is expected for handling large set of data which wasn't possible with traditional databases. Considering the evolution of the industry standards data schema started to change frequently which in turn became a big setback for traditional databases where altering schema is very slow and disruptive. Data storage in Industries like Mechanical or Oil and Gas plays crucial role in the field of data processing and transformation. From the basic inspection report to sensor data, every single entity is captured and processed for analysis, usage of traditional databases for such high load would concur a heavy wait time and cost to the companies. While parquet which came up with industrial revolution of supporting columnar data processing created a big impact in the data engineering domain [1].

Parquet and Python was strongly coupled which made the ETL and Analytics easier than the traditional ones. Parquet reads only the required columns for the query or feature set which increased the performance in big data analytics or machine learning. Parquet used dictionary encoding, run length encoding and bit packing for compressing repetitive values in the file, which resulted in 3-10x reduction in size of the file compared to traditional ones. Parquet well supported alteration of columns without rewriting the entire dataset that helped well in agile experimentation of ML features. Apart from this it stood well in the aspects of scalability by supporting petabytes of data efficiently, supported by various tools across the industry and cost efficient compared to other databases[6].

Few common issues industry was facing with Parquet was it was just a file format, which didn't support atomicity, consistency, isolation, and durability. Concurrent updates of data could lead to data corruption or inconsistency. While large amount of data is split into individual parquet files it becomes costly to manage the metadata. Once file is overwritten it's impossible to rollback due to lack of versioning feature. Parquet is generally batch oriented and not much suitable for streamlining or real-time processing due to requirement of rewriting large file frequently.

Delta-lake was introduced to overcome most of the inconsistencies faced by Parquet file execution. Built on top of Apache Parquet, Delta Lake is an open-source storage layer that enhances big data operations with transactional integrity, consistency, and dependability. It adds time travel, schema enforcement, ACID transactions, and unified batch and streaming functionality to Parquet's effective columnar storage. Delta Lake ensures data accuracy and consistency across dispersed systems while facilitating scalability, concurrent reads and writes through the maintenance of a transaction log, or Delta Log[6].

Delta Lake is a key component of contemporary big data, machine learning, and analytics pipelines since it essentially blends the scalability and cost effectiveness of data lakes with the data management and dependability characteristics of conventional databases.

Delta Lake transforms the traditional Parquet storage model by introducing a sophisticated transaction log system that fundamentally changes how data operations execute. While Parquet operates as a simple columnar file format requiring manual coordination for consistency, Delta Lake builds an entire transactional database layer on top of Parquet files. The core innovation lies in the _delta log directory, which contains a sequence of JSON files that act as an immutable, append-only ledger recording every change to the table. Each transaction—whether it's inserting new data, updating existing records, or deleting rows—generates a numbered commit file (00000000.json, 00000001.json, etc.) that atomically describes the operation. This transaction log serves multiple critical functions such as it provides ACID guarantees through optimistic concurrency control, enables instant metadata discovery without scanning individual files, maintains comprehensive change history for time travel queries, and allows efficient conflict resolution when multiple writers attempt simultaneous modifications.

When Delta Lake executes a read operation, it first reconstructs the current table state by reading the transaction log from the most recent checkpoint, building a complete picture of which Parquet files are currently active and their associated statistics. This approach dramatically outperforms traditional Parquet deployments that must scan potentially thousands of individual file footers to understand the table structure. For write operations, Delta Lake employs a sophisticated two-phase commit protocol: it first writes the actual data to new Parquet files (never modifying existing ones), then atomically commits the transaction by writing a single log entry that makes these new files visible to readers. The system's optimistic concurrency control allows multiple writes to work simultaneously, detecting conflicts only at commit time and automatically retrying failed transactions with exponential backoff.

The real power of Delta Lake emerges during update and delete operations, where traditional Parquet systems fall apart. Instead of the expensive copy-on-write approach that Parquet requires reading entire files, applying modifications in memory, and rewriting completely new files—Delta Lake uses its transaction log to efficiently identify exactly which files contain the target records. It then performs surgical updates by rewriting only the affected files while maintaining perfect ACID semantics through atomic log commits. This architecture enables features impossible with raw Parquet: time travel queries that can access any historical version of the data, schema evolution with automatic compatibility checking, change data capture for incremental processing, and automatic optimization through background processes that compact small files and optimize data layout. The transaction log also provides rich metadata for query optimization, including detailed statistics, partition information, and data skipping indexes, making query planning orders of magnitude faster than scanning individual Parquet file metadata.

In this research, I would like to provide a detailed comprehensive analysis using python data pipelines between Delta Lake and parquet storage formats in big data lake. My research highlights how delta lake advances over parquet in terms of transactional integrity, schema enforcement, real time streaming. This research also provides review of practical benefits that enhance performance and reliability in data intensive applications.

II. BACKGROUND

2.1 PARQUET:

Parquet is a way to store data that makes it fast to analyse large amounts of information. Think of it like organizing a giant spreadsheet in a much smarter way that allows computers to find and process information incredibly quickly.

The fundamental difference between Parquet and traditional data storage lies in how information is organized. Traditional databases store data row by row, like reading a book line by line. If you want to find all the ages in a customer database, you must read every single row even though you only care about one piece of information from each record[3]. This approach works fine for small datasets but becomes painfully slow with large amounts of data. Parquet takes a completely different approach by storing data column by column, like having separate lists for each type of information. When you want to find all ages, you simply read the "age" column directly without touching any other data, making queries dramatically faster[3].

Parquet files are organized like a well-structured filing system with multiple levels of organization. The largest units are called row groups, which are like file folders that hold chunks of data. Within each row group, information is separated into column chunks where similar data types are stored together, such as all names in one section and all ages in another. These column chunks are further divided into pages, which represent the smallest storage units. At the end of each file is a footer that acts like a detailed table of contents, providing a summary of what's inside without having to open and examine everything.

This organizational structure provides several major benefits that make Parquet incredibly efficient. First, it achieves much better compression because similar data stored together compresses more effectively than mixed data. For

example, a column containing ages like 25, 26, 27, 28 can be compressed much more efficiently than rows containing mixed information like names, addresses, and numbers. Second, Parquet enables faster queries by only reading the columns you need and can skip entire sections of data that don't match your search criteria. The format includes built-in shortcuts and statistics that help the system find relevant data quickly without examining irrelevant sections[3].

Perhaps most importantly, Parquet files are self-describing, meaning each file includes comprehensive information about its structure and contents. You always know exactly what type of data you're working with, and the files include detailed statistics like "this section contains ages between 18 and 65" to help skip irrelevant data during queries. This smart query processing capability means that if you're looking for people over 50 years old, the system can automatically skip sections that only contain people under 30 without reading any of that data, loading only the exact columns and rows you need into memory.

To understand the real-world impact, imagine you have a database of one million customer records containing names, ages, addresses, and purchase history. With traditional row-based storage, finding the average age would require reading all one million complete records, forcing you to process names, addresses, and purchase data even though you only need ages. With Parquet, you would read only the age column data, skip any data sections that don't contain ages in your target range, and process only what you need, making such queries ten to one hundred times faster.

This efficiency makes Parquet particularly powerful for analytics, reporting, and data science work where you typically analyse specific columns across large datasets rather than working with complete individual records [3]. Parquet enables reading less data from storage for faster queries, using less memory by loading only what you need, taking less storage space through better compression, and being more reliable through built-in data validation and structure information. The format essentially transforms how we think about data storage, moving from a one-size-fits-all approach to an optimized system designed specifically for analytical workloads.

Below diagram figure 1 showcases the parquet file structure in a clear detailed view on how each ETL job process data from csv change log to parquet tables and how they data warehouses build data dashboards that are useful for powerBi users.

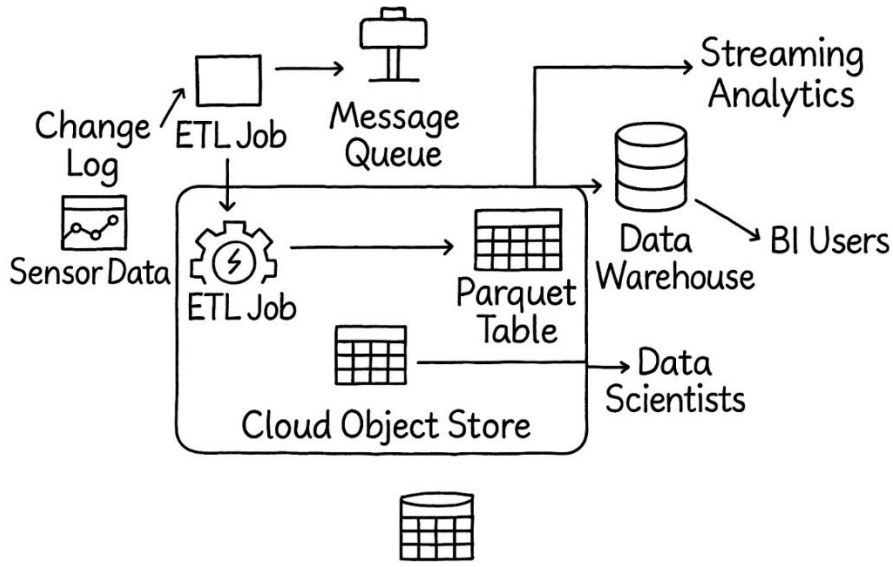

Figure 1: parquet structure

2.2 DELTA LAKE

Delta Lake is Parquet files with a smart tracking system that makes data storage reliable and safe. Regular Parquet files store data efficiently but can't handle multiple people changing data simultaneously or recover from failures. Delta Lake adds a "_delta_log" folder that records every change like a detailed diary tracking who did what and when.

The core innovation is its transaction log system, working like a bank ledger. Before making any changes, Delta Lake writes exactly what it plans to do in numbered journal files (00000.json, 00001.json). This "write-ahead logging" means if something goes wrong, it can either complete the operation or undo it entirely, preventing corrupted data. The system creates periodic checkpoints for quick state summaries without reading every journal entry.

Delta Lake achieves ACID properties - four guarantees for reliable operations. Atomicity ensures operations succeed completely or fail completely by recording changes before execution. Consistency automatically rejects data that violates schema rules or constraints. Isolation lets multiple people work simultaneously by giving each a consistent data snapshot. Durability immediately saves completed transactions to permanent storage, surviving system crashes.

The system uses optimistic concurrency control, assuming people won't conflict until proven otherwise. Instead of locking datasets (which is slow), it allows simultaneous work and checks conflicts only when saving changes. This provides better performance since conflicts are typically rare in data scenarios.

Delta Lake runs on Apache Spark while maintaining Parquet's performance benefits, creating a "lakehouse" architecture that combines data lake flexibility with data warehouse reliability. You get fast analytics and bulletproof data management, including time travel queries to see historical data states.

Figure 2 showcases the deltalake file structure in a clear detailed view and how it is built as a transaction layer on top on paraquet files. This diagram figure 2 also demonstrates how a transaction log called delta log built on source parquet files process into a delta table via schema enforcement and ACID guaranties.

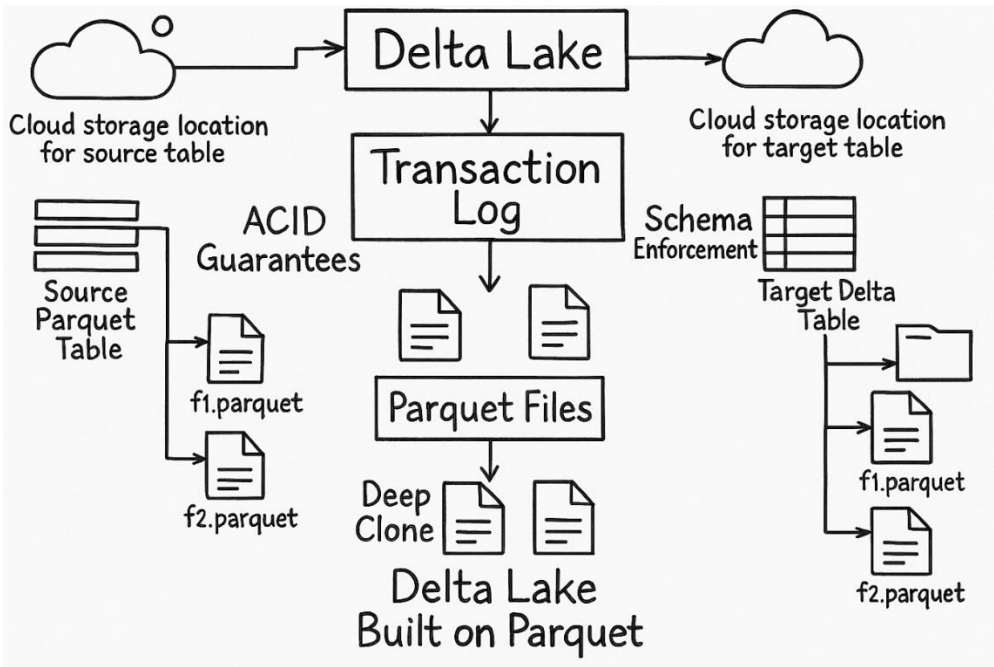

Figure 2: delta lake table structure

2.3 HOW DELTA IMPLEMENTS ACID OVER PARQUET:

Atomicity:

Delta Lake implements atomicity over Parquet files by never modifying existing files and instead using a transaction log to coordinate which files are considered part of the table at any given time. When an operation needs to update data, Delta Lake writes new Parquet files containing the changes while keeping the original files unchanged, then atomically commits the transaction by writing a single JSON log entry that declares which files should be added to or removed from the table[13]. Since Parquet files are immutable once written and the log entry write is atomic at the filesystem level, either the entire transaction succeeds with all new files becoming visible, or it fails with the original Parquet files remaining as authoritative dataset, eliminating any possibility of partial updates or corrupted intermediate states.

Consistency:

Delta Lake enforces consistency over Parquet files by centralizing schema definition and validation in the transaction log rather than relying on individual Parquet file schemas, which can vary and cause compatibility issues. Before writing any new Parquet files, Delta Lake validates that the incoming data conforms to the table's official schema stored in the transaction log metadata, automatically rejecting operations that would create schema conflicts or violate defined constraints. This approach solves Parquet's inherent problem of schema drift across files by maintaining a single source of truth for the table structure and ensuring all Parquet files in the table collection follow the same validated schema and business rules.

Isolation:

Delta Lake provides isolation over Parquet files through versioned snapshots recorded in the transaction log, allowing readers to access a consistent collection of Parquet files even while writers are creating new ones[13]. When a read operation begins, it uses the transaction log to determine which specific set of Parquet files represents the table at that version, creating an immutable view that won't change even as concurrent writers add new Parquet files to subsequent versions. This solves the fundamental problem with raw Parquet directories where readers might see inconsistent combinations of old and new files during concurrent operations, instead guaranteeing that each transaction sees either the complete "before" state or the complete "after" state of all relevant Parquet files.

Durability:

Delta Lake ensures durability over Parquet files by treating the transaction log as the authoritative record of which files constitute the table, making the actual Parquet files recoverable and constructible even if some are lost or corrupted. The transaction log, stored on durable distributed storage, contains complete metadata about every Parquet file that was ever part of the table, including file paths, statistics, and creation details, allowing the system to detect missing files and potentially recover them from backups or replicas. This approach transforms Parquet files from critical single points of failure into constructible data components, since the durable transaction log can always tell you exactly which files should exist and their characteristics, enabling recovery procedures that would be impossible with standalone Parquet files that have no external tracking system.

III. PERFORMANCE COMPARISON

Read Performance:

Parquet delivers excellent read performance for pure analytical queries due to its optimized columnar format, advanced compression, and efficient predicate pushdown capabilities, but suffers from significant metadata discovery overhead that requires scanning file footers across potentially thousands of files before query execution can begin. Delta Lake matches Parquet's columnar read performance once data access begins but dramatically improves query start-up time by maintaining rich metadata and statistics in the centralized transaction log, enabling

instant file discovery and more aggressive predicate pushdown that can eliminate entire file groups without reading any data. For large tables with many files, Delta Lake often achieves 10-100x faster query planning while maintaining identical data scanning performance.

Write Performance:

Parquet provides fast write performance for simple append operations since it only needs to create new files without coordination overhead, making it ideal for batch ETL jobs that write large amounts of data infrequently. Delta Lake introduces slight write overhead due to transaction log maintenance, statistics collection, and conflict detection, typically adding 5-15% to write times for pure append operations, but this overhead is negligible compared to the massive benefits for data reliability and consistency. The performance gap narrows significantly for complex write patterns, and Delta Lake's ability to handle concurrent writes without external coordination often results in better overall system throughput.

Update and Delete Performance:

Parquet suffers catastrophic performance degradation for update and deletes operations, requiring full file rewrites even for single record changes, making small updates prohibitively expensive and often taking hours for operations that should complete in minutes. Delta Lake dramatically outperforms Parquet for modification operations by using its transaction log to identify only affected files and rewriting just those portions, typically achieving 10-1000x better performance for updates and deletes while maintaining full ACID guarantees. Additionally, Delta Lake's change data capture capabilities enable incremental processing workflows that are impossible with raw Parquet, eliminating the need for expensive full table scans to detect changes.

Concurrent Access Performance:

Parquet has no built-in concurrency control, leading to race conditions, data corruption, and the need for external coordination mechanisms that severely limit concurrent access patterns and often require complex application-level locking. Delta Lake excels in concurrent scenarios through optimistic concurrency control that allows multiple readers and writers to operate simultaneously without blocking each other, automatically handling conflicts, and providing consistent performance even under high concurrency loads. The transaction log enables efficient coordination without the performance penalties of traditional database locking, often achieving better throughput than Parquet systems that must serialize operations to maintain data integrity.

Operational Performance:

Parquet requires significant operational overhead including manual file compaction, partition management, and schema coordination across applications, with performance degrading over time as small files accumulate and metadata becomes fragmented. Delta Lake provides automated optimization capabilities including background file compaction, statistics collection, and data layout optimization that maintain consistent performance over time without manual intervention. Features like OPTIMIZE and Z-ORDER commands can dramatically improve query performance, and the centralized metadata management eliminates the operational complexity of coordinating schema changes and file lifecycle management across multiple systems and applications.

Below table 1 demonstrates the detailed performance comparison graph between Parquet and Delta Lake file formats with metric score ranging on a scale of 1-10

Table 1: Performance Metrics Compared

Metric	Parquet Score	Delta Lake Score	Key Insight
Operational Performance	6	9	Delta Lake excels due to ACID compliance and schema evolution.
Concurrent Access Performance	4	8	Delta Lake supports safe concurrent writes via transaction logs.
Update/Delete Performance	2	9	Parquet is immutable; Delta Lake allows efficient upsets and deletes.
Read Performance	8	7	Parquet's columnar format offers slightly faster reads.
Write Performance	7	8	Delta Lake adds overhead but optimizes writes with compaction and indexing.

The below graph as shown in figure 3 visually represents that Delta Lake is superior for dynamic, multi-user environments, with Parquet best fit for static, read-heavy workloads

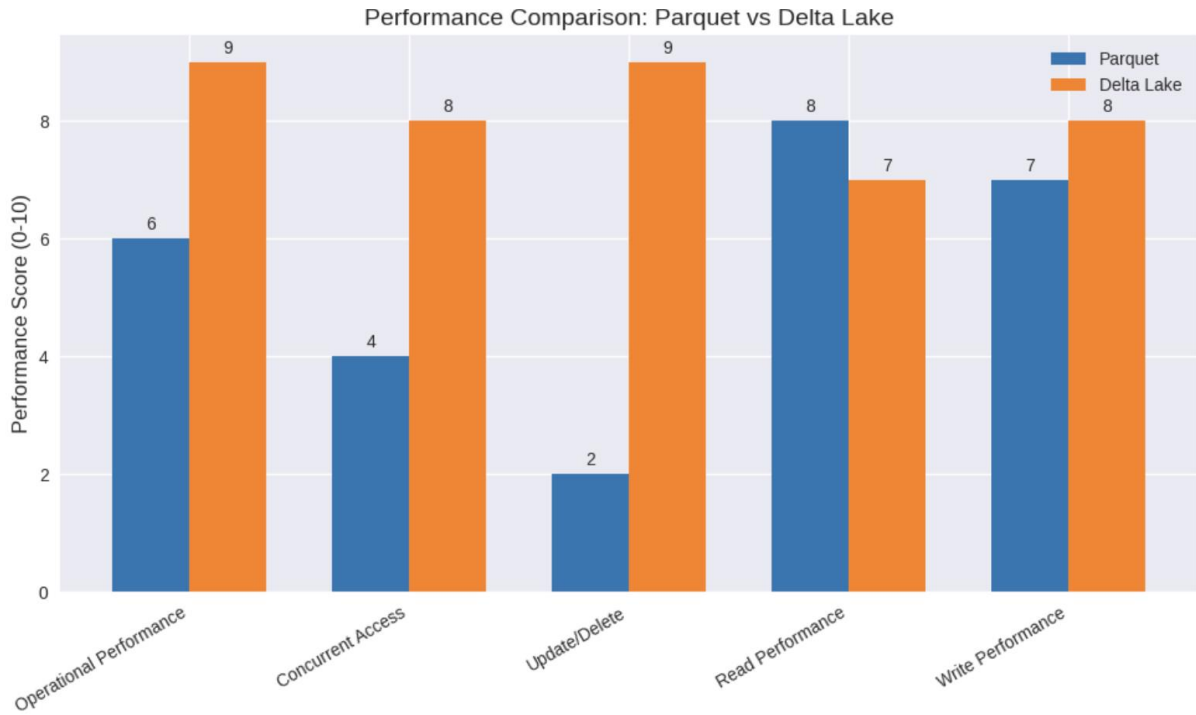

Figure 3: performance comparison delta vs parquet

Below graph as shown in figure 4 also depicts the performance improvement provide by delta lake when compared with parquet for various merge statements of delete, insert and upsert inferred from [5].

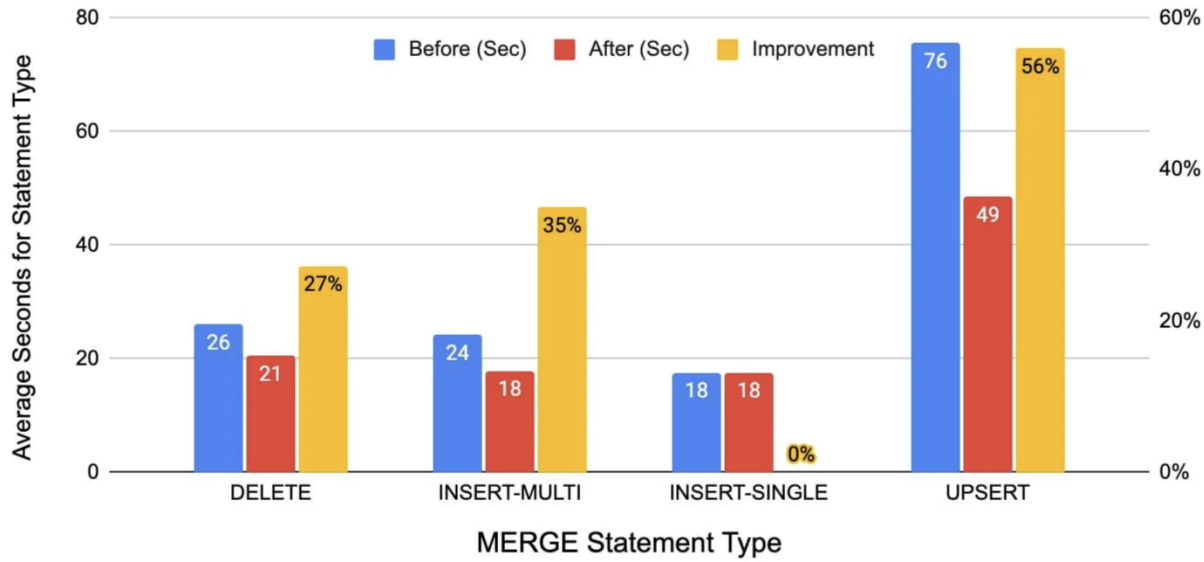

Figure 4: performance comparison delta vs parquet

IV. Implementation

In operations like reporting, historical analysis, and recurring ETL jobs where high throughput is more crucial than instant results, batch processing—which involves managing data in huge chunks that are gathered over time and then processed collectively—is a good fit. Streaming processing, on the other hand, handles data as it comes in real time, providing low-latency insights for applications like live dashboards, IoT monitoring, and fraud detection. While streaming necessitates more intricate handling of state, parallelism, and fault tolerance, batch processing is more straightforward and has historically been associated with formats such as Parquet. To make real-time, streaming applications dependable and consistent in addition to conventional batch processing, Delta Lake expands on Parquet's capabilities by providing ACID transactions and schema enforcement. In our scenario we are going to compare processing of data in batches which is done through native parquet and streaming data through delta log.

4.1 Batch Processing in Parquet

Batch processing refers to the collection and processing of data in large volumes at scheduled intervals rather than continuously in real time. Parquet, being a columnar storage format, is widely used for such workloads because it enables efficient compression, optimized scans, and fast analytical queries over large datasets.

Sample Code:

```
from pyspark.sql import SparkSession


  # Create Spark session
  spark=SparkSession.builder.appName("ParquetBatchExample").getOrCreate()


  # Sample runtime data (simulating a daily batch load)
  batchdata = [
    ("Alice", "2023-09-01", 1200),
```

```
    ("Bob", "2023-09-01", 1500),
    ("Charlie", "2023-09-01", 1000),
    ("Alice", "2023-09-02", 1100),
    ("Bob", "2023-09-02", 1600),
    ("Charlie", "2023-09-02", 1050)
]

# Define schema (name, date, sales)
columns = ["employee", "date", "sales"]

# Create DataFrame
df = spark.createDataFrame(batchdata, columns)

# Write the batch to Parquet (simulating daily ETL job)
df.write.mode("overwrite").parquet("/tmp/sales_batch_data")

# Read the Parquet batch file back
parquet_df = spark.read.parquet("/tmp/sales_batch_data")

# Example: Aggregate sales by employee (batch analytics)
result = parquet_df.groupBy("employee").sum("sales")
result.show()
```

Result –

```
+--------+-----------+
|employee|sum(sales) |
+--------+-----------+
|   Alice|       2300|
|     Bob|       3100|
| Charlie|       2050|
+--------+-----------+
```

Here, sales data for multiple days is collected and written to Parquet in batch mode.

Later, it's read and aggregated to produce insights (e.g., total sales per employee).

This job could run once per day or at another interval, which is the essence of batch processing.

Typical Runtime Ranges

For the small sample dataset I showed (6 rows only):

Write to Parquet: ~1–2 seconds.

Read + Aggregation: ~2–4 seconds.

For realistic big data workloads (millions of rows, GB–TB scale):

Write: 1–10 minutes depending on cluster size and partitioning.

Read + Aggregations: A few seconds to several minutes (much faster than CSV/JSON due to Parquet compression + columnar scans)

4.2 STREAMING DATA USING DELTA

In system operations like reporting, historical analysis, and recurring ETL jobs where high throughput is more crucial than instant results, batch processing—which involves managing data in huge chunks that are gathered over time and then processed collectively—is a good fit. Streaming processing, on the other hand, handles data as it comes in real time, providing low-latency insights for applications like live dashboards, IoT monitoring, and fraud detection[. While streaming necessitates more intricate handling of state, parallelism, and fault tolerance, batch processing is more straightforward and has historically been associated with formats such as Parquet. To make real-time, streaming applications dependable and consistent in addition to conventional batch processing, Delta Lake expands on Parquet's capabilities by providing ACID transactions and schema enforcement inferred from [4].

Sample Code:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StringType, IntegerType

# Create Spark session with Delta support
spark = (
SparkSession.builder.appName("DeltaStreamingExample")
.config ("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension")
.config ("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")
 .getOrCreate()
   )
# Schema for sales data
schema = (
 StructType()
.add("employee", StringType())
```

```
    .add("date", StringType())
    .add("sales", IntegerType())
)

# Stream input (simulate new daily/hourly sales CSV files arriving)
stream_df = (
    spark.readStream
    .schema(schema)
    .option("header", "false")
    .csv("/tmp/stream_sales_input")   # Put CSV files here
)

# Write the stream into a Delta table (real-time ingestion)
query = (
    stream_df.writeStream
    .format("delta")
    .outputMode("append")
    .option("checkpointLocation", "/tmp/delta_sales_checkpoint")
    .start("/tmp/delta_sales_table")
)

# Read the Delta table (real-time query view)
delta_stream_read = spark.readStream.format("delta").load("/tmp/delta_sales_table")

# Example: Aggregate sales per employee in real-time
agg_stream = (
    delta_stream_read.groupBy("employee").sum("sales")
)

# Show results on console
output_query=agg_stream.writeStream.outputMode("complete").format("console").start()

output_query.awaitTermination()
```

Output:

```
-------------------------------------------
Batch: 0
-------------------------------------------
+--------+-----------+
|employee|sum(sales) |
+--------+-----------+
|   Alice|       2300|
|     Bob|       3100|
| Charlie|       2050|
+--------+-----------+

-------------------------------------------
Batch: 1
-------------------------------------------
+--------+-----------+
|employee|sum(sales) |
+--------+-----------+
|   Alice|       3600|
|     Bob|       4500|
| Charlie|       3250|
+--------+-----------+
```

Parquet required one-time write & read. Delta Streaming ingests new data continuously, keeps transactions consistent, and updates aggregates live.

Runtime -

Initial start-up: 15s

Micro-batch 0 (initial historical files): 5s.

Micro-batch 1 (new CSV dropped in): 2s.

Micro-batch 2 (another file): 3s

4.3 COMPARISON PARQUET VS DELTA

Workflow diagram as shown in figure 5, clearly shows how Parquet (Batch Processing) operates in traditional ETL patterns where jobs run once, process complete datasets, and finish - ideal for scheduled reports and historical analysis but providing no real-time capabilities. Delta Lake (Streaming). As shown in figure 5 transforms csv files by enabling continuous micro-batch processing that handles new data as it arrives while maintaining ACID guarantees and automatic recovery, making it perfect for real-time applications like fraud detection and live dashboards. The fundamental difference is that Parquet requires you to choose between batch or streaming architectures, while Delta Lake unifies both paradigms. With Delta, you can run streaming jobs that continuously ingest real-time data while simultaneously running batch analytics on the same tables, eliminating the complexity of maintaining separate systems for different processing patterns[11]. The trade-off is slightly higher start-up latency for Delta streaming (15s vs 5s) due to checkpoint initialization, but this enables much more sophisticated capabilities including exactly once processing, automatic failure recovery, and the ability to serve both real-time and historical queries from the same dataset - something impossible with traditional Parquet-based batch processing.

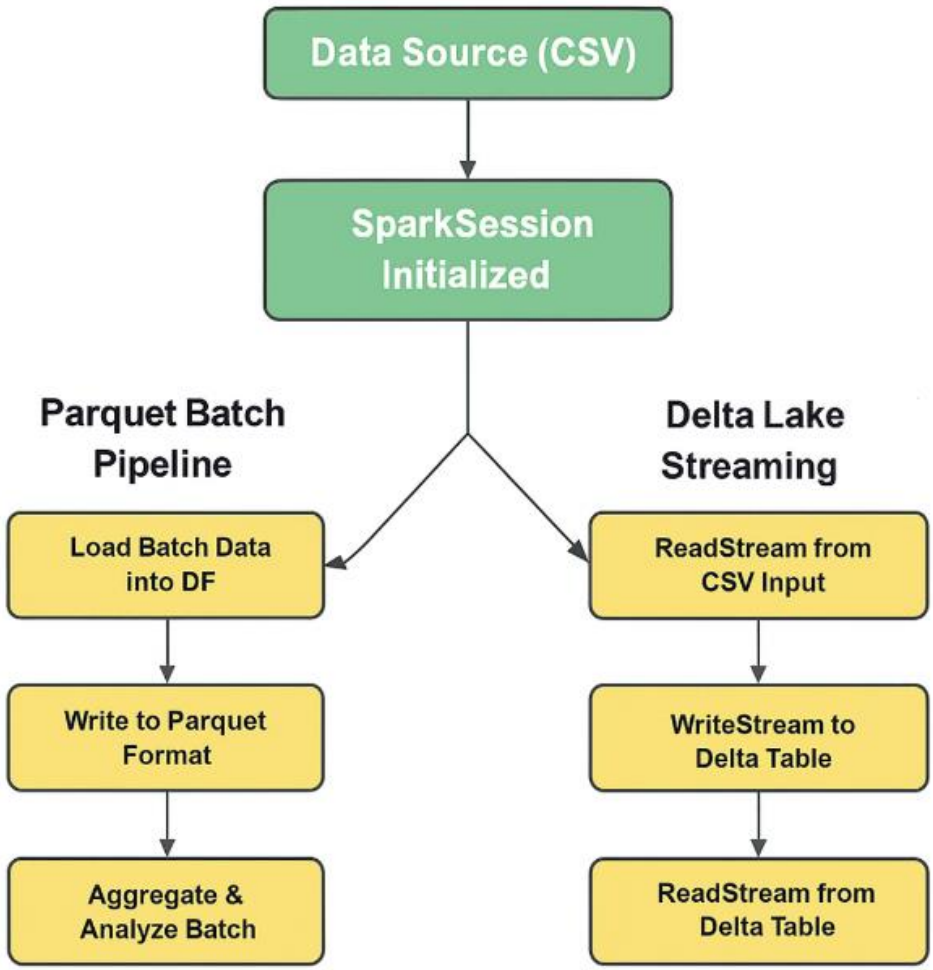

Figure 5: workflow diagram parquet vs delta

Raw data which is exported from transaction data systems form the base for reporting, decision making and analytics. When a pyspark engine is activated on top of these data files in csv format it enables variable data handling across critical clusters used for enterprise grade workloads.

Parquet workflow:

Parquet Batch file as illustrated in figure 5 workflow diagram follows static injection to load data in data factory and there is no support for incremental data update until the existing job complete successfully. In parquet batch processing data quality and freshness depends on batch frequency and could be a risk for real time analytics and time sensitive decision making which may impact business.

Delta lake workflow:

Delta lake provides read stream capability from csv format using continuous ingestion and spark structured streaming for real time data availability. Utilizing ACID properties, schema enforcement and version control deltalake converts the from read stream into write stream delta table with delta logs to do time travel and provide consistent, reliable data even during simultaneous writes. After the data has been converted into write stream delta table it then processes into read stream delta tables with delta logs often used to develop time sensitive dashboards and analytic reports[11]. This contributes to make data driven decisions and with less latency and less error handling improving operations intelligence and live metrics.

Table 2: Comparison table b/w parquet and Delta processing

	Parquet (Batch)	**Delta (Streaming)**
Execution Mode	One-time batch job	Continuous micro-batches (real-time)
Start-up Latency	~5 seconds	~15 seconds (initial setup + checkpoints)
Processing Time	~30s (for complete dataset)	~5s per micro-batch (depends on data size)
Job Completion	Finishes after data is processed	Runs continuously until manually stopped
Data Freshness	Processes historical data only	Processes new data as soon as it arrives
Schema Evolution	Manual	Automatic
Use Cases	ETL jobs, reports, historical analysis	Real-time dashboards, fraud detection, IoT, logs, operational intelligence
Business Agility	Limited	High
Cost efficiency	high	Low due to increased cluster costs.
Fault Tolerance	Limited (requires rerun on failure)	Strong (checkpointing + ACID transactions)

Table 2 clearly demonstrates that business agility is very high for delta streaming with enforced schema evolution, fast processing time and high quality of data freshness. Parquet batch processing is good for sales reporting,

historical data processing, etc whereas delta processing is efficient to reduce latency and build real time data analytics that are time sensitive such as temperature, pressure, etc.

V. USE CASE – DELTA LAKE FORMAT

Banking:

Banking sector stands best as one of the use cases for the use of delta lake as it adds a transactional layer on top of every parquet file for each transaction made and helps in maintaining fraud detection, compliance and banking transactional audit, data governance, risk analysis and customer analytics and reporting.

Delta lake contributes to real-time data sensitive fraud detection systems by allowing financial banks process unified historical batch processing and streaming transaction data with built in single architecture ensuring fraud detecting models are trained for complete intelligence and swift intervention. The ACID properties of delta lake prevent data corruption while a streaming job fails and ignores partial data updates to have consistent data. Change Data feed feature analyses minute row level changes which is crucial in banking sector for audit and helps in reverse tracking the suspicious activity for full history.

In stock market and trading, risk management relays on building predictive models which relay on accurate historic data. Point in time data snapshots provided by delta lake also plays a major role in building financial machine learning models that predict credit risk and market risk.

Delta lake provides efficient data governance by combining audit history and schema enforcement. It also helps build better customer analytics dashboard and operational dashboards ensuring only required data files are queried and skipping unwanted read of data for faster and efficient reliable data delivery.

Oil and gas Industry:

Oil and gas companies widely use delta lake in storing data like sensor data, machinery temperatures from IoT sensors, equipment logs, equipment data, oscillating frequencies, compressor pressures, workorder data, etc that require ordering new equipment and repair parts that needs to be replaced. This in turn result in use of delta lake in time travel to estimate the equipment costs for future maintenance window, analyse pre order of detective parts, operation costs, labour costs and required manpower to complete the maintenance schedule within planned window. Table 2 clearly shows the advantage of delta file usage in oil and gas industry.

Preventive maintenance is one of the key factors in oil and gas industries as failure to analyse and predict machinery wear and tear could lead to significant loss of unplanned downtime resulting in huge financial losses and could incur huge safety risks. Building the preventive machine learning models often require a reliable data source with real time streaming which delta lake can provide capturing sensor data, operational data logs, equipment logs, time series data from rigs, compressors, pumps, oil rigs, temperatures, pressures, and maintenance history to predict system failure well in advances and optimize production losses and recovery rates. Analysis that builds on the above data provided by delta lake helps O&G companies to forecast machinery and equipment failures and plan maintenance window of that machinery or that section of refinery or plant for maintenance without disrupting other production processes.

The knowledge gained form the predictive analysis dashboard on data present in delta lake for oil and gas companies helps build recover strategies and optimized models to ensure maximum oil extraction from wells and rigs with minimum cost investments and manpower required.

My utilizing predictive analysis dashboards created oil and gas companies can create reliable Environmental social governance (ESG) reports, prevent environmental accidents and auditing for regulatory compliance.

Warehouse management:

Delta lake plays a vital role in the warehouse management systems with it inherit ACID transactions, data versioning, schema enforcement to improve data reliability, performance, and quality by building data lakes called lake houses. In warehouse management delta lakes offer unified data platform with streaming data to reduce maintenance of separate lake for their systems. Delta lake offers high performance and easy access to data without duplication.

It becomes easy to time travel and track the history of a product that is helpful during warehouse audit. Delta lake provides optimized data schema providing quick warehouse analytics on large datasets that bring data lake performance closer to data warehouses.

Big giant companies like Amazon, flipcart, Shien all of them use delta lake as their data lake to manage their warehouse inventory.

VI. Limitations / offsets of Delta Lake:

For delta lake as real time clusters need to run actively 24x7 to target the organisational SLA and KPI's, there is a 45-60 % increase in infrastructure costs inferred from [8] compared to batch processing and is not cost efficient.

Delta lake real time pipelines are more complex and becomes difficult to debug in terms of state management, water marking for late data and no downtime schema validation. This process involves more time and slows the development productivity of data engineers.

Big Querying or snowflake inserts cost additional premium cluster costs when compared to bulk processing as costa are calculated from cluster runtime and storage I/O as inferred from [9].

Batch processing delivers higher throughput stability due to its ability to process large backlogs in one run ignoring traffic spikes while delta lake experience load when traffic spikes increase that led to temporary lag and requires tuning of Sparks parallelism to avoid performance degradation.

As realtime streaming depends on checkpoint integrity, small misconfiguration can lead to data handling issues, errors and increase additional costs.

VII. Conclusion

The study conducted in this paper demonstrates the key differences that distinguish parquet and delta lake in modern world data engineering process design. Despite parquet set its standard as an efficient solution for columnar data storage and compression, its use in mission critical workloads is restricted and its drawbacks to support ACID compliance, schema enforcement and streaming data flow as it is a batch processing and does not support parallel real time operations. To overcome these short comes of parquet, delta lake has been built as a transactional layer on top of parquet to provide unified batch streaming, schema evolution, parallel read/write operations, and time travel to track history.

According to our investigation, Parquet continues to deliver exceptional performance in batch-driven analytical workloads where high compression efficiency as well as affordability are critical. Delta Lake performs noticeably better by ensuring consistency and scalability in situations that require for reliability, parallel execution, and real-time insights—for example, large-scale warehouse management, oil and gas predictive maintenance, or banking fraud detection. The implementation examples demonstrate how Delta Lake streamlines operations by automating metadata computation, simplifying query design, and enabling streaming ingestion without compromising data integrity.

Delta Lake enables enterprises to develop "lake house" architectures that integrate the flexibility of data lakes with the reliability of data warehouses, providing operational and business value beyond performance.

In conclusion, Delta Lake shouldn't be considered of as an alternative for Parquet. Rather, it should be viewed of as a natural extension that makes Parquet a more powerful, reliable, and sustainable storage format. Organizations which utilize Delta Lake have advantages like long-term reliability, streamlined data operations, and the ability to use both historical and current data sources to develop useful insights. Further research can broaden this analysis through contrasts Delta Lake to newer options like Apache Iceberg and Hudi or by evaluating how effectively it functions in broader cloud-native environments like Microsoft Fabric and Databricks. Through analysis on cost sensitivity, operational resiliency, competitive advantage by faster data delivery, industry standard SLA's and KPI's, monitoring sophistication and should be considered before making decision on what framework to choose between parquet and delta lake rather than relying on technical innovation.

REFERENCES:

1. Amazon Web Services. (2018). (Amazon Web Services) From data lakes to rivers of insight: Our vision for the oil & gas industry and AWS partnership [eBook]. https://d1.awsstatic.com/Industries/Oil/AWS_Data_Lakes_eBook_O%26G_Final.pdf
2. Datanexum. (2025). Boost oil and gas operations with data analytics for ERP in 2025. https://datanexum.com/insights/f/boost-oil-and-gas-operations-with-data-analytics-for-erp-in-2025
3. Lang, L., Hernandez, E., Choudhary, K., & Romero, A. H. (2025). ParquetDB: A lightweight Python Parquet-based database [Preprint]. arXiv. https://arxiv.org/pdf/2502.05 (1)311
4. Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., Ghodsi, A., Stoica, I., & Zaharia, M. (2018). Structured streaming: A declarative API for real-time applications in Apache Spark. SIGMOD '18: Proceedings of the 2018 International Conference on Management of Data, 13–25. https://people.eecs.berkeley.edu/ (1) (1)~matei/papers/2018/sigmod_structured_streaming.pdf
5. Harris, J. (n.d.). Delta Lake performance. Del (Harris)ta Lake Blog. https://delta.io/blog/delta-lake-performance/
6. (1) (2020). Delta Lake: High-performance ACID table storage over cloud object stores. Proceedings of the VLDB Endowment, 13(12), 3411–3424. https://doi.org/10.14778/3415478.3415560 [1]
7. Micheal, L. (2024). Trade-offs between batch and real-time processing: A case study of Spark Streaming in enterprise data pipelines. [Unpublished manuscript].
8. Karau, H., & Warren, R. (2017). High performance Spark: Best practices for scaling and optimizing Apache Spark. O'Reilly Media. (1)
9. Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., Ghodsi, A., Stoica, I., & Zaharia, M. (2018). Structured streaming: A declarative API for real-time applications in Apache Spark. Proceedings of the VLDB Endowment, 11(12), 1910–1922.
10. Kodakandla, P. (2023). Real-time data pipeline modernization: A comparative study of latency, scalability, and cost trade-offs in Kafka-Spark-BigQuery architectures. [White paper].
11. Salim, H. P. (2025). A comparative study of Delta Lake as a preferred ETL and analytics database. International Journal of Computer Trends and Technology (IJCTT), 73(1), 65–71. https://doi.org/10.14445/22312803/IJCTT-V73I1P108

12. Delta Lake. (n.d.). *Build lakehouse with Delta Lake*. Retrieved September 13, 2025, from https://delta.io/ (Lang)

13. Delta Lake. (2024). *Structured Spark streaming with Delta Lake: A comprehensive guide*. Retrieved September 13, 2025, from https://delta.io/blog/structured-spark-streaming/