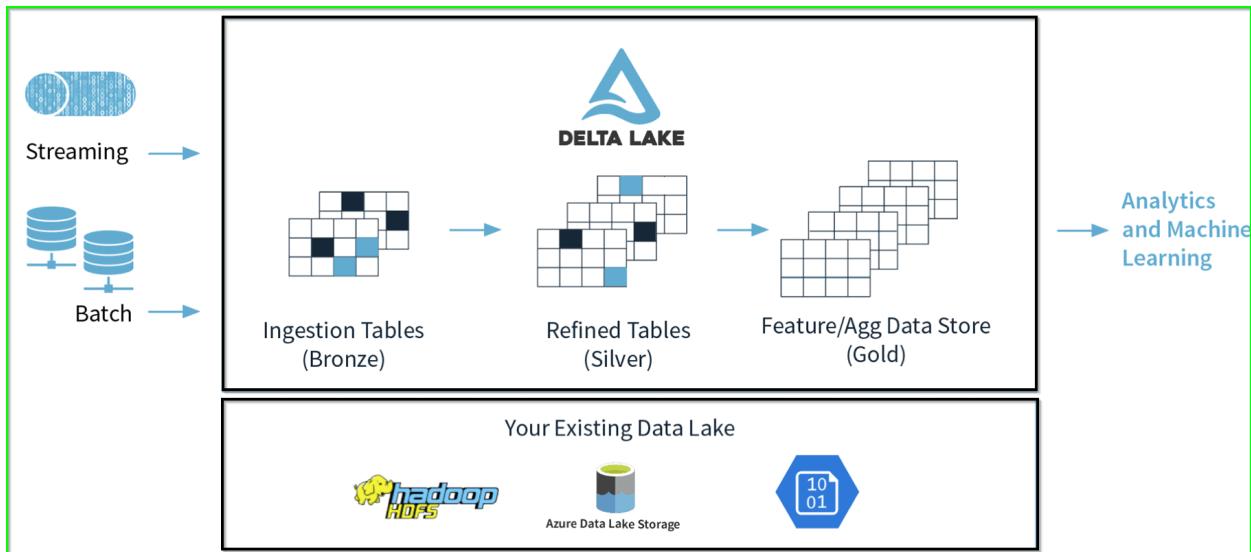


## Delta Lake Architecture: Project implementation:

This project utilizes the Medallion Architecture to manage data quality and reliability. In this implementation, local storage is used to save all data directly on the machine.



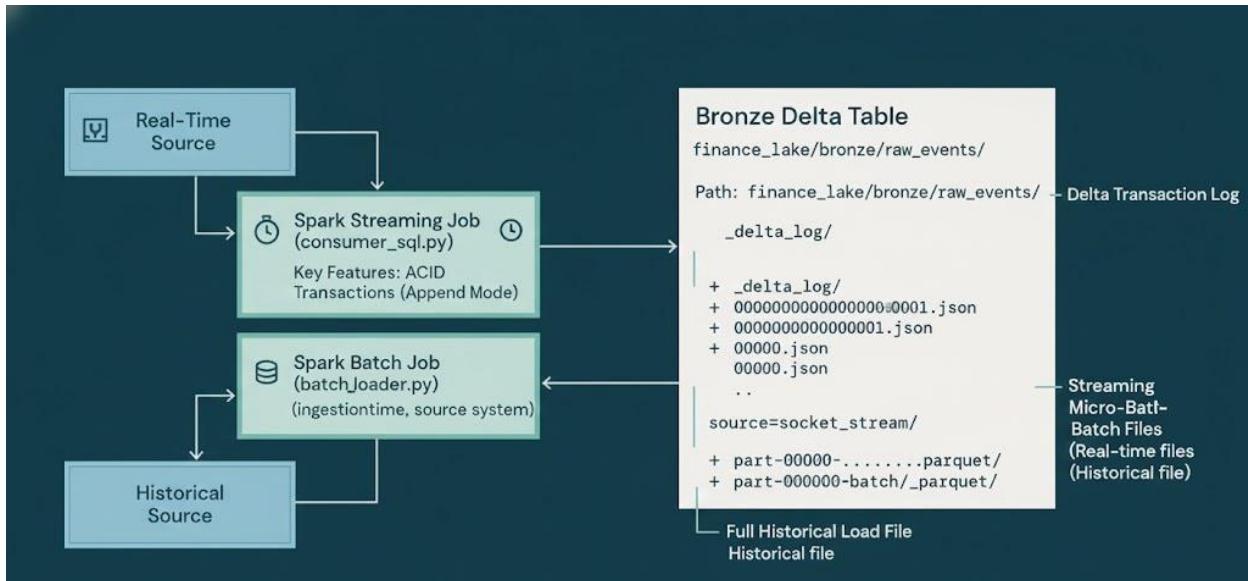
### 1. The Bronze Layer: Unified Ingestion and the data sources :

#### A. Data Sources and Ingestion (The Workers)

In our system, data arrives in two different ways:

- **Historical Data (Batch - JSON File):** This is the old data that already exists, such as past stock prices. It arrives all at once in a large file.
- **Real-Time Data (Streaming - Live Stock Prices):** This is the live data that arrives continuously, such as current stock price updates in real time.

Our system is designed to handle both past data and real-time data at the same time.



## B. Data Processing with Spark (How the Data Is Treated)

We use Apache Spark to process the data with two different jobs:

- **Batch Job (`batch_loader_delta.py`):** This job loads the historical data once and stores it into the system.
- **Streaming Job (`consumer_sql.py`):** This job listens continuously to the real-time data stream and inserts new data as it arrives.

✓ This allows our system to work with both static and live data efficiently.

## C. The Bronze Delta Table (The Secure Vault)

Both workers place their materials into the exact same final storage unit. This is the **Bronze Delta Table (`finance_lake/bronze/raw_events`)**.

- **Unified Raw Data:** This means the past truckloads and the current conveyor belt packages are all organized in the same place.
- **ACID Transactions (The Security Guard):** This is the **Delta Lake**'s superpower. It acts as a security guard ensuring two things:

1. **Safety:** If both workers try to put materials in at the same instant and there's a problem, the security guard guarantees no messy, half-finished box gets left behind. (The write either fully succeeds or fails cleanly).
2. **Order:** It keeps a perfect log of everything that goes in, making it a reliable archive.

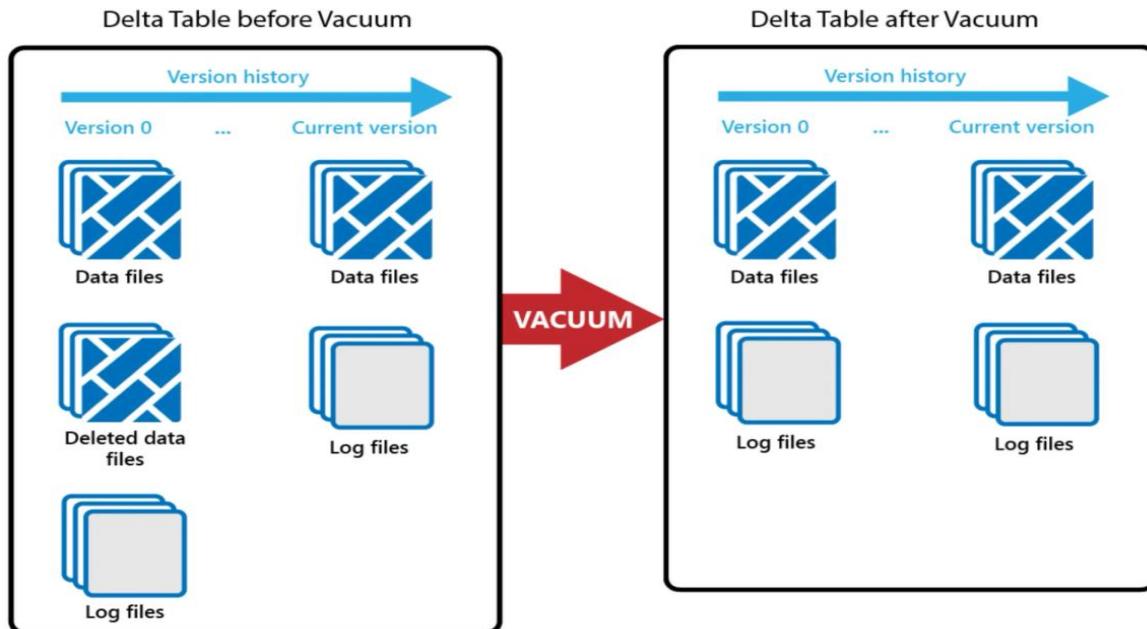
#### D. Maintenance: Data Cleaning (`delta_maintenance.py`)

The **VACUUM** operation is a maintenance task that cleans up the physical storage by deleting files that are no longer needed for the current table state or for Time Travel.

##### 1. The Components

The Bronze Layer is composed of two key elements that the VACUUM job interacts with:

- **Delta Transaction Log** (`_delta_log/`): This holds the history of every transaction (write, update, delete) and lists which Parquet files are **ACTIVE** for each version of the table.
- **Data Files** (.parquet): These are the physical files that hold your raw stock data.



##### 2. The Process

The **VACUUM** command follows two strict criteria to decide which files to delete:

- **Identify Stale Files** : Delta finds all Parquet files that are not listed in the current or recent versions of the Transaction Log (File is Unreferenced (Logically Deleted))
- **Check Retention** : Delta checks the timestamp when the file was logically deleted. If this timestamp is older than the specified retention period, the file is eligible for permanent removal (file is older than x hours/days ).

## 2. Refined Tables (Silver Layer)

The Silver layer represents the core refinement stage of the lakehouse architecture. Its role is to transform raw, semi-structured financial events from the Bronze layer into a **clean, standardized, deduplicated, and enriched dataset**. This layer produces high-quality data that is suitable for **analytical consumption, real-time enrichment, and downstream advanced processing**.

### 2.1. The Silver Pipeline (`silver_pipeline.py`)

The `silver_pipeline.py` script implements a **Spark Structured Streaming pipeline** that continuously consumes data from the Bronze Delta Table and applies a sequence of validation, transformation, feature engineering, and **deep learning-based inference** steps before persisting the results into the Silver Delta Table.

The pipeline relies on the **`foreachBatch`** processing model, which provides precise control over each micro-batch and enables the execution of **transactional Delta Lake MERGE operations**, ensuring correctness, consistency, and fault tolerance in a streaming environment.

#### Key Processing Steps

##### 1. Data Conformation and Cleaning

###### **Primary Key Generation (`event_id`):**

A deterministic and globally unique primary key (`event_id`) is generated for each financial event using a cryptographic SHA-256 hash over the combination of symbol, timestamp, and price. This identifier is essential for reliable deduplication, late-event handling, and idempotent updates.

###### **Data Quality Filtering:**

Records containing invalid values, such as non-positive prices,

are filtered out to ensure the integrity and trustworthiness of the Silver dataset.

#### **Audit and Lineage Tracking:**

A metadata column (`silver_processing_time`) is added to capture the timestamp at which each record is processed in the Silver layer, enabling traceability and auditability across the data pipeline.

### **2. Deduplication Strategy Using Delta Lake MERGE**

A primary responsibility of the Silver layer is to guarantee **exactly-once processing semantics** in a continuous streaming context. This is achieved through a two-stage deduplication strategy:

#### **Intra-Batch Deduplication:**

Within each micro-batch, Spark window functions (`row_number` partitioned by `event_id`) are used to remove duplicate events arriving in the same batch, retaining only the most recent record.

#### **Inter-Batch Deduplication (Transactional MERGE) :**

The deduplicated micro-batch is merged into the Silver Delta Table using `event_id` as the join condition.

- **WHEN MATCHED UPDATE:** If a record with the same `event_id` already exists, the stored record is updated with the latest values.
- **WHEN NOT MATCHED INSERT ALL:** If the `event_id` does not exist, the record is inserted as a new entry.

This atomic upsert mechanism leverages **Delta Lake's ACID guarantees**, ensuring consistency, reliability, and correctness under continuous data ingestion.

### **3. Feature Engineering for Time-Series Analysis**

To enrich the financial events with temporal context, the pipeline performs time-series feature engineering:

#### **Lag-Based Features:**

Previous prices are computed using the `LAG` window function, enabling the calculation of **log returns**, a standard metric used to quantify relative price movements in financial markets.

#### **Rolling Sequences:**

A rolling window is applied per financial symbol to construct ordered sequences of recent log returns. These sequences capture

short-term temporal dynamics and provide contextual input for sequential modeling.

#### **4. Deep Learning Inference Integration (GRU-Based)**

As part of the Silver processing workflow, a **deep learning inference phase** is integrated directly into the streaming pipeline:

- A **Gated Recurrent Unit (GRU)**-based inference function is applied to rolling sequences of log returns.
- GRU models are well suited for modeling sequential and time-dependent data, making them appropriate for financial time-series analysis.
- The inference step produces a continuous predictive signal (gru\_prediction) for each event, enriching the dataset with forward-looking information.

By embedding GRU-based inference at the Silver layer, the architecture enables **real-time, model-driven data enrichment**, ensuring that downstream layers consume data that already contains predictive signals.

#### **5. Silver Data Persistence**

The final enriched dataset—containing cleaned records, engineered features, and deep learning predictions—is persisted as a **Delta Lake table** in the Silver zone. Schema evolution is enabled to allow seamless extension of the dataset as new attributes or model outputs are introduced.

#### **2.2. Silver Verification (`verify_silver.py`)**

The `verify_silver.py` script is used to validate the correctness and integrity of the Silver layer.

##### **Deduplication Validation:**

The script reads the Silver Delta Table in batch mode and groups records by the primary key (`event_id`). By counting the number of occurrences per key, it verifies that no duplicates exist.

A result showing **zero duplicate keys** confirms that the deduplication logic and Delta Lake MERGE operations are functioning correctly and that the Silver layer maintains strict data quality guarantees.

### 3. Feature and Aggregated Data Store (Gold Layer)

The Gold layer represents the final, consumption-oriented stage of the lakehouse architecture. It is designed to consume curated data from the Silver layer and provide **business-aligned, analytics-ready datasets** tailored to specific reporting or analytical use cases.

#### 3.1. The Gold Pipeline (`gold_pipeline.py`)

The Gold pipeline is conceptually responsible for reading the enriched Silver data and applying **domain-specific aggregations and transformations** to prepare datasets optimized for downstream consumption, such as dashboards, reporting tools, or analytical applications.

At this stage, the Silver layer serves as the trusted source of truth, containing clean, deduplicated records and enriched features, including predictive signals generated during Silver processing. The Gold layer builds on this foundation to expose data in a form that aligns with business and analytical requirements.

### 4. Feature/Agg Data Store (Gold Layer) Consumption

The Gold Layer is the final stage of the Medallion Architecture, providing highly aggregated and business-ready data optimized for specific consumption patterns, such as analytics dashboards. The successful pipeline validates the layer's readiness for predictive analytics and operational auditability.

#### 4.1 Functional Dashboard

A functional dashboard was implemented using **Streamlit** to directly consume the feature-rich Gold Delta Table, showcasing the immediate utility of the final feature set.

**Operational Controls**

Delta Data Version (Time Travel) ?

Current (V1) ▼

**Financial Predictive Insights**

Running `load_data(...)`.

RUNNING... Stop Deploy ⋮

**Operational Controls**

Delta Data Version (Time Travel) ?

Current (V1) ▼

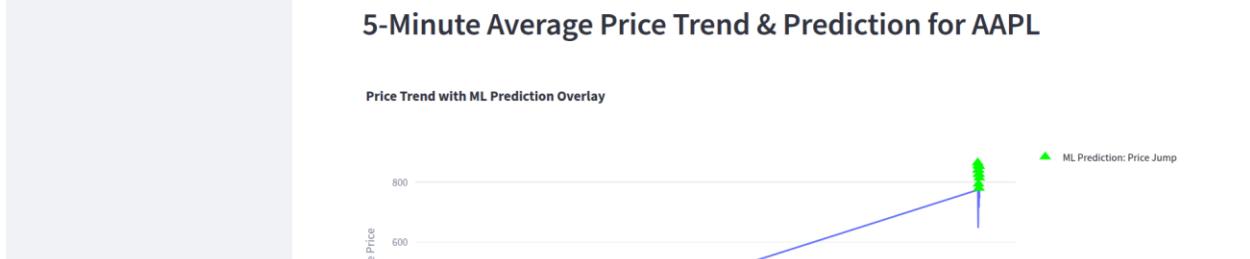
Filter by Stock Symbol

AAPL ▼

**Financial Predictive Insights**

Loading CURRENT DATA: Latest version of the Gold Table.

Model Accuracy (Test Set) 77.78% Current 5-Min Prediction No Jump (0.0) Total Records Loaded 175 (Gold Layer) Deploy ⋮



The screenshot shows a user interface for a machine learning model audit. On the left, there's a sidebar titled "Operational Controls" with dropdown menus for "Delta Data Version (Time Travel)" set to "Current (V1)" and "Filter by Stock Symbol" set to "AAPL". At the top right, there's a "Deploy" button and a three-dot menu icon. The main area is titled "Recent Predictions & Actual Outcomes (Audit)". Below the title is a table with columns: end, avg\_price, prediction, and label. The table contains 10 rows of data, each representing a prediction made at a specific time (end) with an average price, a predicted outcome (prediction), and an actual outcome (label). The data is as follows:

	end	avg_price	prediction	label
34	2025-12-04 22:00:00	745.3573	0	0
33	2025-12-04 21:55:00	752.6431	0	0
32	2025-12-04 21:50:00	767.2727	0	0
31	2025-12-04 21:45:00	774.1437	0	0
30	2025-12-04 21:40:00	814.178	1	0
29	2025-12-04 21:35:00	852.6384	1	1
28	2025-12-04 21:30:00	762.5805	0	0
27	2025-12-04 21:25:00	826.5833	1	1
26	2025-12-04 21:20:00	715.0234	0	0
25	2025-12-04 21:15:00	779.5946	1	1

- **Model Accuracy:** Displays the reliability of the predictive model (e.g., \$77.78\%\$ accuracy) for the analyst<sup>4</sup>.
- **Actionable Signals:** The main chart plots the **Average Price** and overlays green markers using the **prediction** column<sup>5</sup>. This visual turns complex machine learning output into a simple, **actionable trading signal**<sup>6</sup>.
- **Audit Detail:** A table allows users to audit the model's past performance by comparing the actual outcome (label) against the model's predicted outcome (prediction)<sup>7</sup>.

## 4.2 Operational Reliability: Delta Lake Audit

Two critical audits confirm the data integrity and operational reliability of the entire data lakehouse architecture.

### A. Data Quality Proof (Silver Layer)

The integrity of the data consumed by the Gold Layer is guaranteed by the Silver Layer's deduplication strategy.

- **Strategy:** The Silver Pipeline uses the Delta Lake **MERGE** command with **event\_id** as the primary key. This atomic **Upsert** logic ensures the Silver table remains clean and free of redundant events.
- **Result:** The verification script confirms the MERGE operation is functioning correctly by verifying that no group has a count greater

than one. This provides definitive proof of **zero duplicates** remaining in the Silver table.

#### **B. Time Travel Proof (Gold Layer)**

The Gold Layer proves full audibility of the feature store, a key requirement for financial systems.

- **Mechanism:** The pipeline demonstrates this capability by executing a query to read a previous version of the table using `option("versionAsOf", last_version - 1)`.
- **Audit Use Case:** This enables the analyst to instantly load and investigate the **exact set of features** used for model training at any historical point, fulfilling the requirement for auditabile and restorable data.
- **Proof Point (Pipeline Log):** The successful log output confirms this auditability:

Reading Version 2 of the Gold Table (Before last overwrite):

Count in Previous Version (2): 175