



Lakehouse Streaming for Intelligent Decision Systems

Benjamin Clark

Asst Professor

Abstract

Real-time data engineering pipelines play a pivotal role in acquiring, processing, and analyzing streaming data to support intelligent decision-making systems. Taking a practical approach, this work examines real-time data pipelines within the Azure Databricks platform using PySpark. It presents a set of foundational concepts alongside a flexible implementation framework to guide the design of real-time data technology workloads. Empirical breadth and repeatability are demonstrated through educational data sources and case studies that showcase working applications of real-time technologies for monitoring and controlling business processes — including anomaly detection in streaming financial indicators and dynamic pricing strategies.

Intelligent decision systems draw on theories and technologies from a range of disciplines, including Artificial Intelligence (AI), Machine Learning (ML), Data Mining, Decision Support Systems (DSS), and Operations Research (OR). Within the Digital Business and Technology domain, real-time data engineering pipelines address the acquisition, processing, and analysis of streaming data to enable data-driven decisions grounded in a company's current business environment. Notably, most intelligent decision systems emphasize the data-driven component of decision-making more than the decision engine itself. Building on this, the following discussion centers on a Digital Business and Technology use case for designing an intelligent decision system — integrating both data engineering and data science aspects of data-driven decision-making — and implementing a series of real-time data technology workloads in Azure Databricks using PySpark.

Keywords : Real-time data pipelines, Azure Databricks, PySpark, streaming, online feature store, micro-batching, anomaly detection, decision systems.

1. Introduction

The second half of this information-dense article provides insights into best practices and the Databricks implementation of intelligent management systems leveraging streaming data. A generic framework for real-time data engineering pipelines, supported by diverse case studies, forms the core of the discussion. These pipelines are vital for information systems that inform and support agile, intelligent decisions with readiness for implementation through software packages.

Intelligent decision systems are key components of a large proportion of real-world business scenarios. These systems revolve around the principles of effect-driven decision acting on information that is as up-to-date as possible or strategic planning based on real-time forecasts. Forecasts, in turn, rely on data that is gathered during operations and processed in near real time. The information may then be published for fast use or incorporated into online feature stores. Online feature stores can be leveraged to reduce exploitation costs and detect anomalies in the processes. Azure Databricks provides the necessary environment for building the data pipelines and their supporting features.



Real-time data engineering pipelines feed information systems that inform and support agile, intelligent decisions. These are action-oriented, risk-hedging systems that require data ingestion and processing capabilities that work at sufficient speeds. Following these principles ensures that the information is as up to date as possible, thereby supporting a decision-oriented business model. Such systems are characteristic of dynamic pricing, inventory management, and demand forecasting applications.

1.1. Background and Significance

Data have become the most valuable asset of organizations in a rapid digital transformation era. The data explosion is being amplified by the Internet of Things (IoT), which is steadily increasing the velocity of data processing. Data engineering research and development have been mainly focused on batch pipelines, wherein data are ingested, processed, and made accessible for analysis in discrete time intervals. OLAP systems have been primarily employed for executive dashboards and historical analysis, yet many operational applications, such as dynamic pricing and inventory management, require continuous data flow and near real-time decision making. Intelligent decision systems (IDS) are being developed to automatically estimate future states and activate actions based on the predictions.

1.2. Scope and objective

Real-time data pipelines extract, transform, and deliver data for analytical workloads with strict time constraints. The concept forms the foundation for building intelligent decision systems, which provide current recommendations, detect anomalies, recalculate pricing based on supply and demand, and facilitate several other functions. Such systems require decision-making features; therefore, secondary data from streaming or micro-batched pipelines is passed to online feature stores for transformation, combination, and enrichment. Micro-batching on a Stream-Process-Stream architecture pattern further improves decision quality.

Most theories and concepts are generalized and abstract. Nevertheless, concrete implementations on cloud platforms—especially Azure Databricks with its PySpark support for streaming and micro-batching—provide good empirical insights. Real-time anomaly detection in financial data illustrates the concept with a labeled dataset; the trading signal serves as a classification label. A second case study employs Airflow to change stock prices from a Public API based on purchase volume and remaining shelf life, enabling a micro-batch processing architecture and allowing for decision-making features to be added.

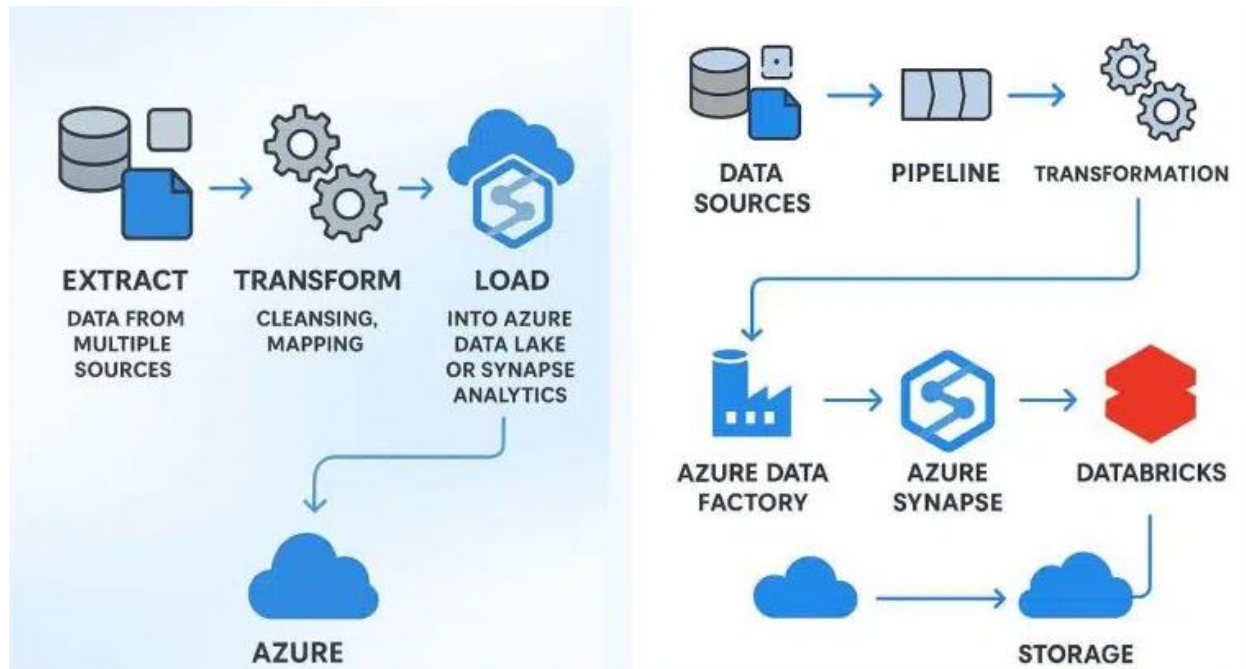


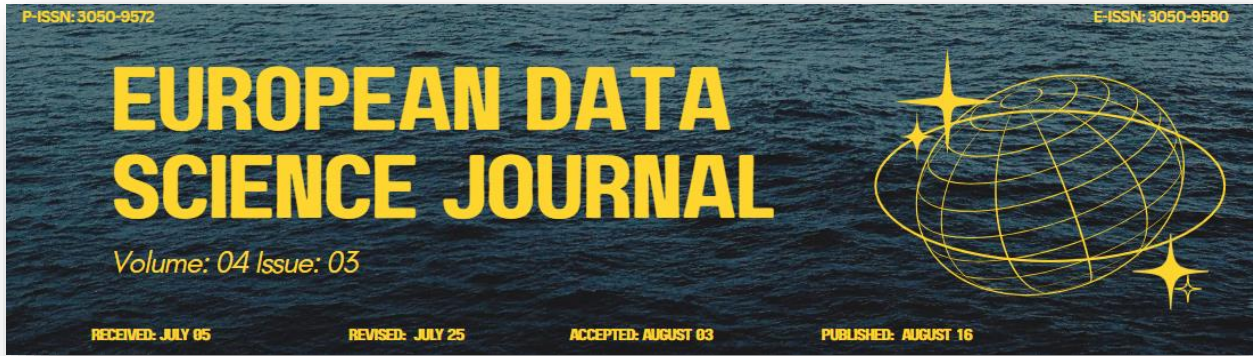
Fig 1: Azure Data Engineering

2. Foundations of Real-Time Data Engineering

With the explosion of Internet-of-Things (IoT) devices and an ever-increasing demand for information systems that enable intelligent decisions, a growing number of data engineering pipelines work with streaming data. These systems produce and analyze information as it arrives, making it possible for organizations to learn about events, detect patterns, and make decisions in real time. Anomaly detection and forecasting for dynamic pricing or inventory management are some areas where timely data is critical for optimal performance. Real-time decision-oriented data logic carefully aligns the timing of data ingestion, processing, and consumption.

Various definitions for streaming data exist, but the defining characteristic is that it arrives in small, incremental pieces and must be processed quickly. Streaming data engineering pipelines usually adopt an architecture pattern for data-driven systems, supporting key requirements for end users through real-time analytics. The focus here is on pipelines deployed on the Azure Databricks platform with PySpark. Though the details of individual production jobs differ, they all share a common design for real-time data ingestion and preprocessing, creation of features within an online feature store, and derivation of target variables. Systems that are designed with these reusable components can easily adapt to different applications and use cases.

Equation 1: Micro-batch throughput equation



Let:

- N_t = number of records arriving in batch window t
- Δt = batch duration in seconds
- R_t = throughput in records per second

By definition, throughput is:

$$R_t = \frac{N_t}{\Delta t}$$

Step-by-step derivation

1. Rate always means:

$$\text{rate} = \frac{\text{quantity}}{\text{time}}$$

2. Here the quantity is the number of records processed in one micro-batch:

$$\text{quantity} = N_t$$

3. The time consumed by that batch window is:

$$\text{time} = \Delta t$$

4. Therefore:

$$R_t = \frac{N_t}{\Delta t}$$

Interpretation

If 24,000 events arrive in a 120-second window, then:

$$R_t = \frac{24000}{120} = 200 \text{ records/sec}$$

2.1. Streaming data concepts

Streaming data refers to continuous flows of data that, in contrast to static datasets, remain in-transit for only a short time. These data flows must be dealt with as soon as possible, before being lost. In principle, even highly dynamic datasets such as live video/audio or stock price quotes are real-time data. Most real-time data-processing tasks, however, deal with information that is

not necessarily live but is still relevant or time-sensitive. Such flows come from sensors that constantly emit telemetry data, such as log events or credit card transactions, or, more generally, events that are generated by an online information system or service.

Data streams exist in an infinite space of possible values over a particular time period, but the vast majority of the values appear only once and for a very brief moment, such that there is essentially only one copy of each value in the entire stream. Thus, most operations on data streams can be performed on increasingly smaller subsets with a minimum of accuracy loss while the processing time shrinks. The required accuracy, or acceptable trading-off between accuracy and latency, is hence the most significant parameter of real-time data-processing tasks. Where resources allow, rapid computation over all possible data must be carried out. The resources required for streaming jobs are then generally estimated by benchmarking a best-effort procedure and dividing the resulting performance by the required accuracy. This benchmarking-and-scaling approach is especially effective, general, and inexpensive in cloud environments that provide on-demand instance provisioning.

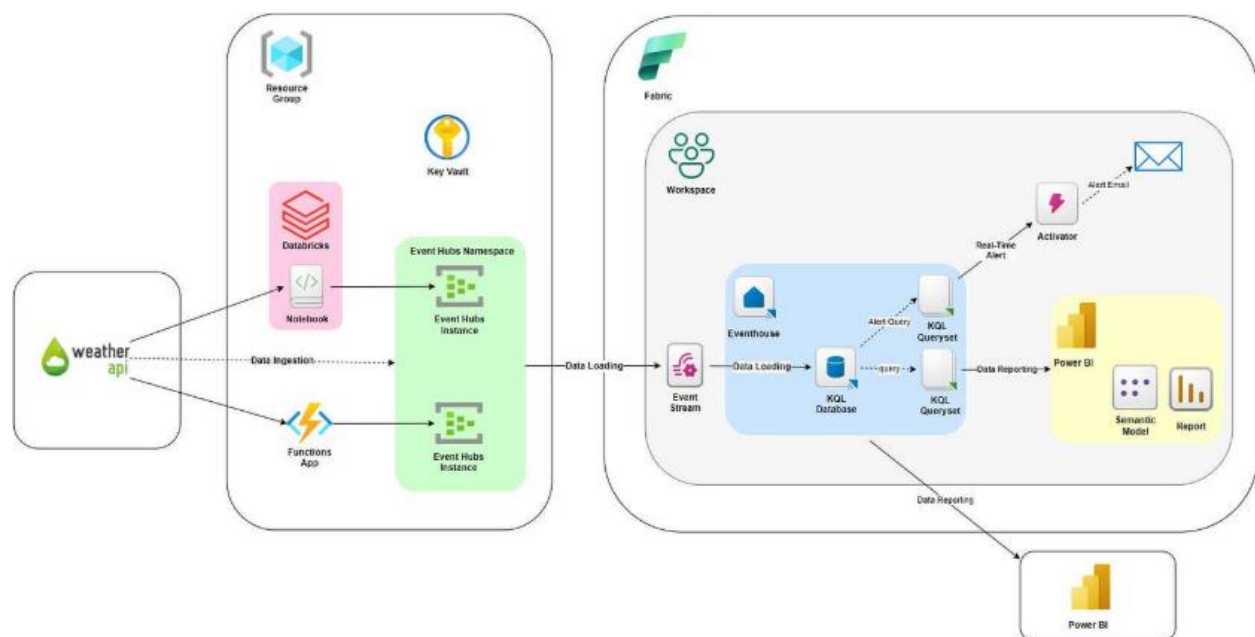


Fig 2: Real-Time Streaming Data Pipeline with Azure

2.2. Architecture patterns for real-time pipelines

Several architectural designs are widely discussed in the literature, with a few examples highlighted here. The simplest architecture is to push the online streaming data into an online storage location as fast as possible, perhaps into an object storage system or a transaction log. These raw data can then be read via batch processing or micro-batch processing from a fault-tolerant storage, fed into a feature extraction job, and stored into an online feature store at a lower frequency for serving prediction requests.



A typical use case for this architecture is a fraud or anomaly detection system in finance. Transactional data can be ingested raw into an object storage like AWS S3 or Azure ADLS and subsequently loaded into a dedicated database during non-peak hours for ad-hoc investigations by analysts. However, in this scenario, non-systemic anomalies must be detected in real time and must trigger immediate responses, so prediction models must be kept up-to-date. Raw streaming transactional data can be continuously fed into a model training job that runs periodically (e.g., every hour). The trained model is then redeployed into an online model store. Predicted results are looked up and read in batches from the model store into a rules engine or a cloud function that checks for results above a given threshold. When a rule triggers, a dedicated system (e.g., AWS SQS, Azure Service Bus) can be populated to initiate the countermeasures.

A more sophisticated architecture collects a wide variety of data sources, such as traditional game-changer and risk factors from social media, traditional news, and non-traditional news and events, all in real time. From those, indicators of new rules (anomalies) can be generated and stored in a feature store to feed into prediction models in an automated cycle. Using these organizations, outside the infrastructure boundaries, a classified looped feedback of successes and failures can be kept and supervised by human entities to notionally improve score accuracy.

Stage	Description	Tools/Technologies	Purpose
Data Ingestion	Collect streaming data from sources	Kafka, Event Hub, IoT Hub	Capture real-time events
Data Processing	Transform & clean incoming data	PySpark (Structured Streaming)	Prepare usable data
Feature Extraction	Generate features in real-time	Online Feature Store	Improve ML predictions
Storage	Store processed data	ADLS, Cosmos DB	Enable retrieval & analysis
Model Inference	Apply ML models	Databricks ML / PySpark	Generate predictions
Action/Decision	Trigger alerts/actions	Azure Functions, Rules Engine	Intelligent decision-making

Table 1: Real-Time Data Pipeline Framework



3. Azure Databricks and PySpark in Real-Time Workloads

Azure Databricks provides an optimal cloud environment for running feature engineering and other data-preparation workloads on streaming data. The platform supports PySpark APIs for streaming data processing, enabling the construction of streaming data pipelines and processing of streaming data frames. Built-in checkpoints, multiple source connectors, and integrated monitoring tools streamline the implementation of real-time data pipelines. Using the fuzzy matching built-in, PySpark sets and manages matched columns automatically, simplifying the matching logic.

Real-time data pipelines are tested, validated, and demonstrated on Azure Databricks using feature-extraction tasks for visualization, online feature stores, and general-inference workloads for intelligent decision systems. Two applications exemplify the abilities of real-time data pipelines: anomaly detection in financial datasets using a pseudo-cloud and intelligent pricing in a competitive e-commerce environment where competitors' prices and inventory levels are scraped at regular intervals.

3.1. Overview of Azure Databricks platform

Azure Databricks provides an as-a-service platform for developing big-data applications with the open-source Apache Spark analytics engine. By integrating Microsoft Azure services, including clusters, resource management, data storage, security, and authentication, it simplifies the construction of Spark-based intelligence applications and devOps. Building on these capabilities, the platform natively supports real-time data pipelines, offers connectors for streaming data sources, implements PySpark APIs for streaming workloads, and provides a micro-batching mechanism.

Azure Databricks supports quasi-real-time or near-real-time workloads by implementing Spark engines. Quasi-real-time data processing supports systems that can tolerate delays in data updates from minutes to hours, such as near-real-time stock-trading systems built on up-to-date but not strictly real-time stock data. Major Azure services, including Azure Data Lake Storage (ADLS), Cosmos DB, Event Hub, and Azure SQL Database, provide support for streaming data and are readily available to Azure Databricks users.

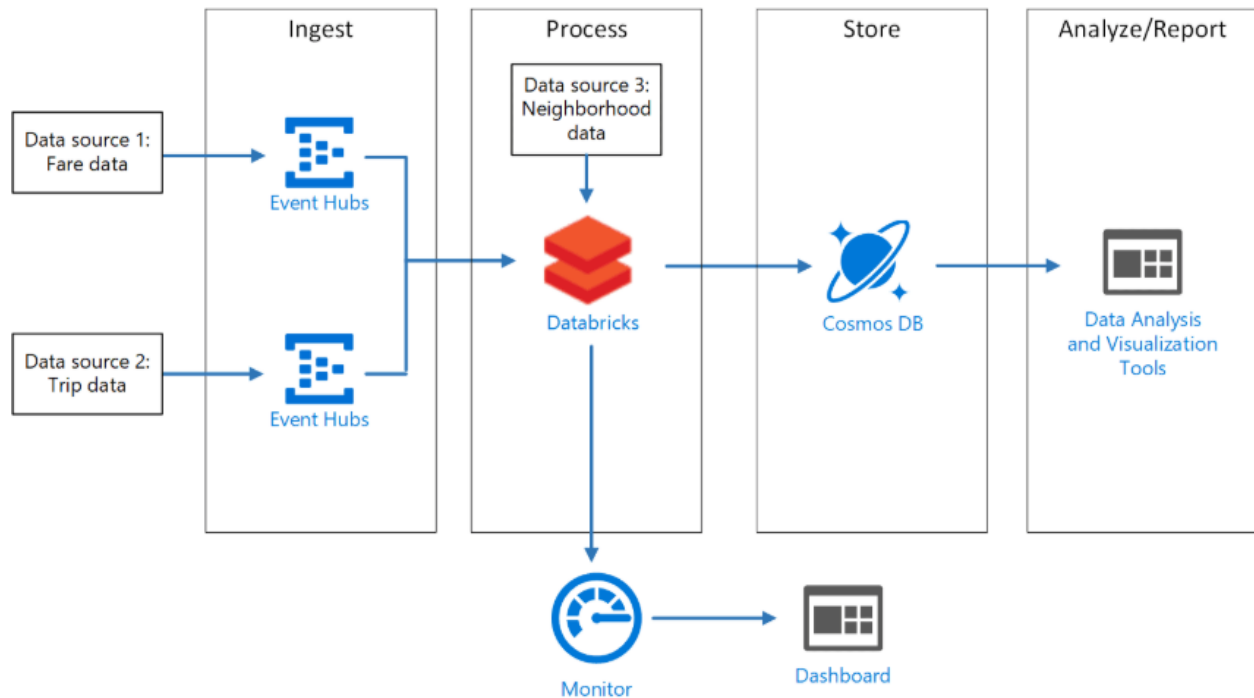


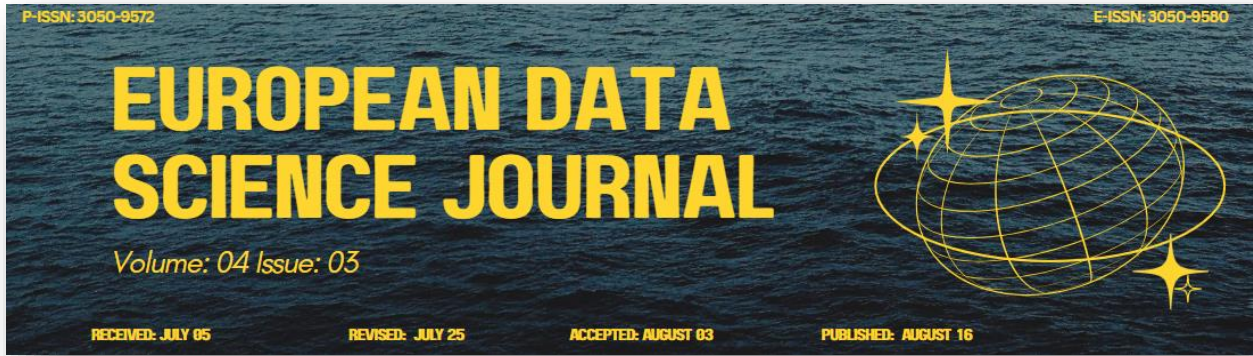
Fig 3: Overview of Azure Databricks

3.2. PySpark APIs for streaming

This section presents PySpark APIs for creating real-time data engineering pipelines and implementing streaming preprocessing modules that run as continuously active jobs and services while ensuring robustness and fault tolerance, with a focus on Azure Databricks as a seamless integrated platform.

PySpark provides two sets of APIs for processing streaming data. The lower-level APIs with the name `StreamXXX` allow defining operations on a single `DStream` abstraction, supporting all Spark operations. These direct operations make it possible to apply user-defined transformations in a streaming context. The higher-level APIs with the name `readStream` and `writeStream` operate on streaming data sources and sinks and are recommended for most users. Operations on a `DataFrame` or `Dataset` associated with a stream result in a stream that can be written to a sink and defines a new streaming query. These higher-level APIs support the same processing guarantees provided by Spark's batch models. Internally, the higher-level operations translate the different stages of the query plan and query execution into sets of operations that are applied to each micro-batch.

The APIs allow specifying options such as the `checkpointLocation` for fault tolerance, employing a streaming sink for continuous output, and choosing an output mode, e.g. `append`, `complete`, or `update`. A monitoring interface shows the current status of all active streaming queries and enables stopping them.



Equation 2: End-to-end latency equation

Let:

- T_{ingest} = ingestion time
- T_{proc} = processing time
- T_{write} = sink/write time
- L = total latency

Then:

$$L = T_{\text{ingest}} + T_{\text{proc}} + T_{\text{write}}$$

Step-by-step derivation

1. End-to-end latency is total elapsed time from source arrival to sink availability.
2. That total time is made of three serial stages:
 - ingest
 - transform/process
 - write/serve
3. For serial stages, total time is the sum:

$$L = T_1 + T_2 + T_3$$

4. Replacing the stages with pipeline terms:

$$L = T_{\text{ingest}} + T_{\text{proc}} + T_{\text{write}}$$

Interpretation

If ingestion takes 10 s, processing 85 s, and writing 5 s:

$$L = 10 + 85 + 5 = 100 \text{ s}$$

4. Designing Intelligent Decision Systems with Real-Time Data



Two core requirements for deploying intelligent decision systems are data ingestion and preprocessing and feature extraction with an online feature store. Both of these operations must be executed in real time so that intelligent decision systems are simultaneously fed with the most recent data and the feature vectors necessary to make decisions. Moreover, in some applications, such as dynamic pricing and inventory optimization, feature engineering is not a simple task and requires human knowledge or advanced specialized techniques. Therefore, these feature extractors can play a key role, as they not only receive real-time input but also implement complex transformations.

To ingest streaming data, Azure Databricks supports various sources and connectors, including many popular protocols and formats: Apache Kafka, Azure Event Hubs, Amazon Kinesis, TCP sockets, files, and others. These connectors allow ingesting incoming data from the cloud, external data lakes, or data published by third-party services, among others. Two types of ingestion jobs can be created: streaming jobs, which use continuous processing for low-latency ingestion and output of events, and near-real-time jobs that use micro-batching. In such jobs, incoming data is collected noncontiguously within controllable time windows where micro-batches are being processed. For processing latency, select a smaller window duration, and for throughput select a larger one.

Azure Databricks provides a flexible and powerful ecosystem for ingesting streaming data from a wide range of sources and connectors, including platforms such as Apache Kafka, Azure Event Hubs, and Amazon Kinesis, along with TCP sockets and file-based inputs. These integrations enable organizations to capture real-time data from cloud systems, external data lakes, and third-party services efficiently. Azure Databricks supports two primary types of ingestion jobs: streaming jobs and near-real-time jobs. Streaming jobs rely on continuous processing to achieve low-latency data ingestion and immediate event output, making them suitable for real-time analytics use cases. In contrast, near-real-time jobs use a micro-batching approach, where incoming data is grouped into small, noncontiguous time windows for processing. This method provides flexibility in balancing performance, as smaller window durations reduce processing latency, while larger windows improve overall throughput.

4.1. Data ingestion and preprocessing

The first two modules in every intelligent decision-making system involve the ingestion and initial encoding of available business context data from disparate sources and formats. The batch or near-real-time characteristics of these activities mean that the focus is not on processing speed but on resource use. Resource management becomes crucial for the third module, responsible for gathering and processing streams of business event data emitted from the context data reservoir, transaction systems, and connected devices as the events occur. The app-enabling infrastructure within Azure Databricks makes this micro-batching framework highly flexible.

In the first case study, an event-based application to sense an alarm in a collection of financial time series classifies them as normal or not normal and emits Basic Attention Tokens as alerts. The application allows adapting placement and retention prices for a set of airplane tickets in the second case study, by monitoring ticket prices on specific routes and identifying periods of unusually high demand.

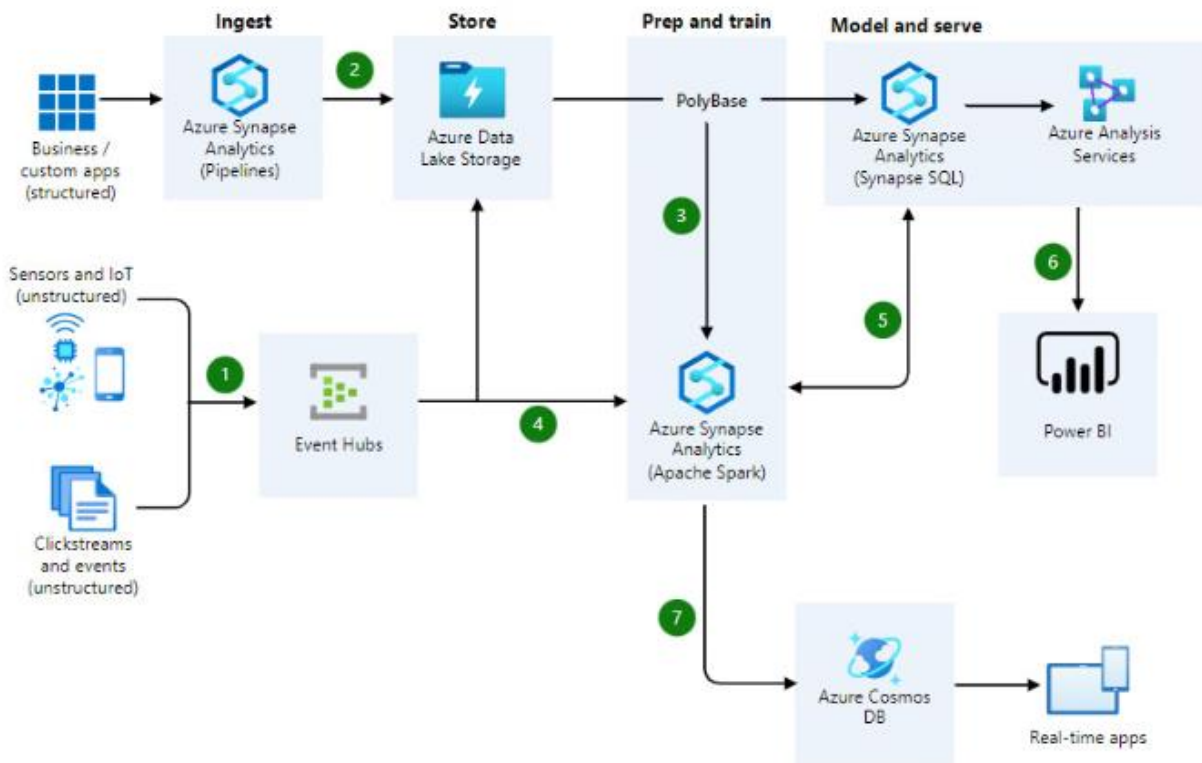


Fig 4: Guide To Real-time Data Ingestion

4.2. Feature extraction and online feature stores

Intelligent decision systems based on event data often need additional information related to the data generated and ingested. In such scenarios, naive modeling may result in low-quality predictions. For instance, the same offline machine-learning model that predicts if a sensor will malfunction may not predict the same from two different sensors from the same environment with the same data. An intelligent way of solving this problem is to create features based on the timestamp, such as the hour of the day or day of the week, or to extract features with autocorrelation lag of N within the prediction time window.

In real-time systems, these features should be computed in an online manner since the prediction is also real-time based. Multiple solutions are available to perform online computations of these features with different latencies supported, depending on the underlying requirement. One of the available solutions is to store the features in an online feature store for later retrieval in other streaming jobs. These online features are generally of three types: (1) Aggregator features, where the features are computed using aggregators like sum, average, min, max, count, first, last, and collecting list; (2) nearest-neighbor features that can be mapped to a machine-learning context to enrich the model with incoming data; and (3) modeling features that use ECS-like predictions as features.



Feature	Batch Processing	Streaming Processing
Data Handling	Large chunks	Continuous flow
Latency	High	Low
Use Case	Historical analysis	Real-time decisions
Processing Mode	Periodic	Continuous / micro-batch
Example	Reports, dashboards	Fraud detection, pricing

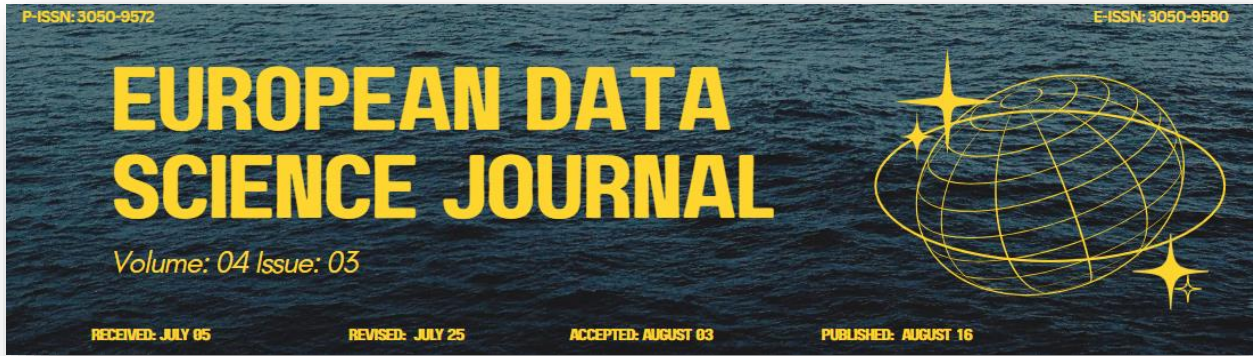
Table 2: Streaming vs Batch Processing

5. Implementation Framework on Azure Databricks

The framework for building and deploying real-time data engineering pipelines on Azure Databricks embraces the platform's offerings, facilitating the rapid deployment of solid proofs of concept (PoCs). A complete animal shelter dataset is ingested through a low-code solution, enabling the demonstration of key streaming concepts via PySpark, followed by a simplified proof of real-time anomaly detection in time-series financial data with custom data sources and connectors. The third phase showcases an internal dynamic pricing model built with batch data along with a full-proof-of-concept Databricks Online Feature Store. Completion of these three steps provides a solid foundation on the platform that allows the author to showcase key aspects within a limited time frame while enabling a deeper exploration of their area of expertise.

The concrete implementation focuses on data sources, connectors for custom streaming jobs, considerations for how data ingestion can benefit from micro-batching even in non-streaming workloads, and data engineering steps that are critical for decision systems but do not possess their own streaming aspect. The other critical pieces—anomaly detection in time-series data and dynamic pricing using Databricks Online Feature Store—connect to the appropriate sections. The main objective is the mapping of real-time data pipelines for intelligent decision systems, so the focus remains on establishing viable Azure Databricks implementations while adhering to these requirements.

Equation 3: Rolling aggregate feature equation



A standard rolling average feature for a stream x_t over a window of size w is:

$$\bar{x}_t = \frac{1}{w} \sum_{i=0}^{w-1} x_{t-i}$$

Step-by-step derivation

1. Suppose the latest observations are:

$$x_t, x_{t-1}, x_{t-2}, \dots, x_{t-(w-1)}$$

2. The sum of those observations is:

$$\sum_{i=0}^{w-1} x_{t-i}$$

3. The average equals total sum divided by number of terms:

$$\bar{x}_t = \frac{\sum_{i=0}^{w-1} x_{t-i}}{w}$$

4. So the rolling mean feature is:

$$\bar{x}_t = \frac{1}{w} \sum_{i=0}^{w-1} x_{t-i}$$

Interpretation

For $w = 4$ and values 9,11,13,15 with current time t at 15:

$$\bar{x}_t = \frac{15 + 13 + 11 + 9}{4} = \frac{48}{4} = 12$$

5.1. Data sources and connectors

Various data sources and connectors can be used to build real-time data engineering pipelines in Azure Databricks. The Streams and Files connector in Azure Databricks is helpful for ingesting streams of real-time data. Azure Databricks provides managed connectors for commonly used cloud storage to read and write from different cloud storage services for data lakes. Azure Blob Storage or Azure Data Lake Store connections can be used for data-lake type of storages. Low-latency data stores for transactions, like Azure Cosmos DB or updates from NoSQL stores, can be used for fast reads and writes with multiple design



considerations and benefits. Any other type of No-SQL stores or data sources that support streaming capabilities or can be polled for data and ingested to Azure Databricks are suitable.

Kafka is a popular message-broker platform for real-time ingestion and dispatch of streaming data in many systems, hence Databricks provides Databricks Connectors for Kafka for integrations. The Kafka source can consume messages in a stream, whereas the Kafka sink can produce messages in a stream. The spark-kafka-0-8_2.11 and spark-sql-kafka-0-10_2.11 packages provide Kafka integration with Spark for consuming and producing data. For the purpose of package installation, the Spark Structured Streaming interface and the appropriate version of a Hadoop distribution supporting Amazon S3 can be used for creating streaming queries using Kafka connectors.

Apache Kafka has become a widely adopted message-broker platform for handling real-time data ingestion and distribution across modern data systems. To support seamless integration with streaming workflows, Databricks offers Kafka connectors that enable both consuming and producing data streams. Specifically, Kafka can act as a source to ingest continuous streams of messages into Spark, and as a sink to publish processed data back into Kafka topics. Integration with Apache Spark is facilitated through packages such as spark-kafka-0-8_2.11 and spark-sql-kafka-0-10_2.11, which provide the necessary APIs for structured streaming operations. By leveraging Spark Structured Streaming along with a compatible Hadoop distribution that supports Amazon S3, users can build scalable, fault-tolerant streaming pipelines that efficiently process and store real-time data using Kafka connectors.

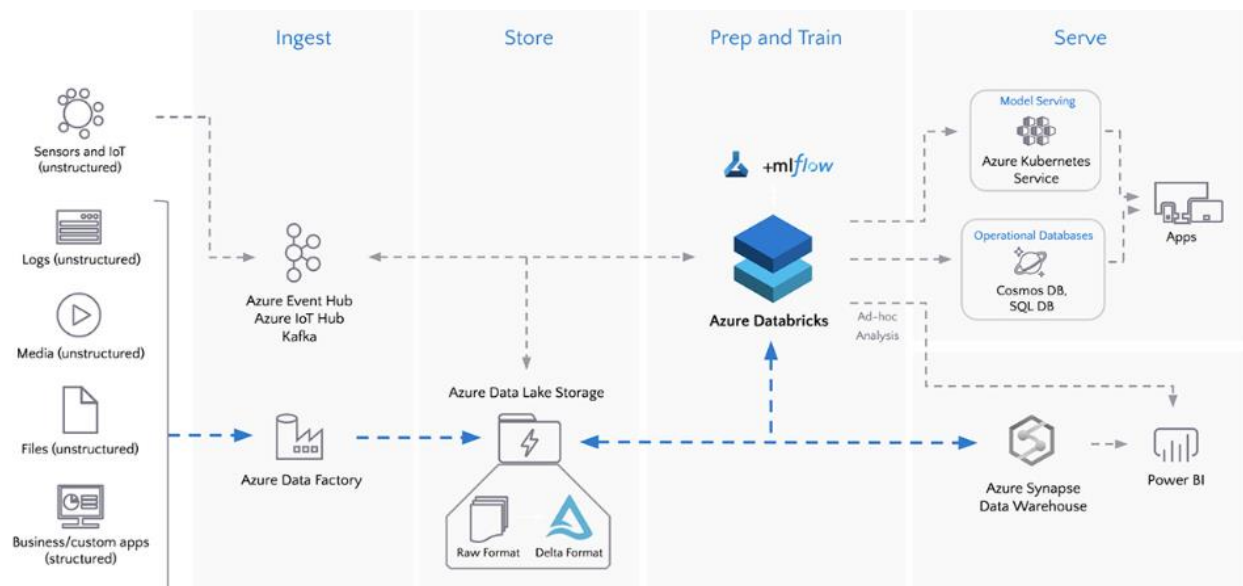


Fig 5: Data Sources to Data Lake and Databricks



5.2. Streaming jobs and micro-batching considerations

Several projects using public data sources employed real-time data engineering concepts on Azure Databricks. As a cloud-based platform, Azure Databricks offers built-in connectors to Azure Event Hub, Azure IoT Hub, Azure Blob Storage, and data lakes such as Azure Data Lake Storage Gen2. Real-time features in Azure Databricks also provide an option for micro-batching, where data latency is reduced by writing data in smaller data blocks. This feature is backed by Lambda architecture, supporting batch and streaming jobs with improved resource availability. For micro-batching, Sigma compute is added to run workloads at a lower cost than regular workloads.

Using micro-batching, Azure Databricks jobs were created on public data sources in the area of anomaly detection, dynamic pricing, and inventory management. Streaming data sources push messages to Azure Event Hub in near real time. Windows-based operations are performed in two-minute intervals.

Architecture Type	Description	Example Use Case
Raw Stream → Storage → Batch	Store raw data, process later	Data lakes
Stream → Model → Rules Engine	Real-time prediction & alerting	Fraud detection
Multi-source Streaming	Combine social, news, IoT data	Risk analysis
Stream-Process-Stream	Continuous pipeline loop	Dynamic pricing

Table 3: Architecture Patterns for Real-Time Pipelines

6. Case Studies and Empirical Demonstrations

The previous section outlined an implementation framework for real-time data engineering pipelines on Azure Databricks. This section presents two empirical use cases of intelligent decision systems based on Azure Databricks. The first case addressed real-time anomaly detection for financial data, and the second explored dynamic pricing and inventory optimization for the tourism industry. A financial transaction dataset with three labeled anomalies was used for the first study, and tourist booking requests for holidays in a popular destination over three years formed the second dataset.

Real-Time Anomaly Detection in Financial Data



Detecting fraudulent credit card transactions is a time-critical task as fraudulent transactions usually have short persistence properties. The overall detection performance can be improved by minimizing the time required to take corrective actions. A labeled dataset produced by the European Credit Card Fraud Detection Challenge is used, containing one day of actual transactions from European credit cards marked as either legitimate or fraudulent. The analysis focuses on a four-hour period with three labeled anomalies. To mimic a real-time working environment, the evaluation matches the time-stamped transactions using a slowly-moving window. All processing steps, including data retrieval from Microsoft Azure Blob Storage, feature extraction, model inference, and anomaly tagging, are implemented as micro-batching streaming jobs with a three-minute batch interval. A fourth job is defined for computing the performance metrics, which runs every 30 seconds.

Equation 4: Lag/autocorrelation feature equation

The simplest lag feature is:

$$f_t^{(N)} = x_{t-N}$$

A common derived change feature is:

$$\Delta_N x_t = x_t - x_{t-N}$$

Step-by-step derivation

1. A lag- N feature means “use the value from N time steps ago.”
2. If current value is x_t , then the lagged value is:

$$x_{t-N}$$

3. So define:

$$f_t^{(N)} = x_{t-N}$$

4. Often we want not just the lagged value, but how much the signal changed relative to that lag.
5. Change = current minus previous:

$$\Delta_N x_t = x_t - x_{t-N}$$

Interpretation

If $x_t = 120$ and $x_{t-3} = 100$, then:

$$f_t^{(3)} = 100, \Delta_3 x_t = 120 - 100 = 20$$

6.1. Real-time anomaly detection in financial data

The recent combination of the Shiny application framework with PySpark enables developers to create interactive applications and dashboards on Apache Spark clusters, either locally or on cloud providers. Shiny allows for live querying of Spark tables, offering filters, aggregators, and even graphical functions. To demonstrate the Azure integration, a simple financial dataset is published on a Databricks Delta table and made available through a Shiny app. Users can dynamically filter and visualize the data. A new Databricks integration extends this to a complete interactive application hosted in Azure, presenting a sample application for anomaly detection in time-series data. The service enables users to expose streaming queries for live dashboarding and reporting and verify detected anomalies against visual inspection. Multiple front ends can interact with the same streaming tables.

Financial Time-Series data serves as the basis for testing. The dataset consists of time series over 7 financial indicators, sampled at 1–10 second intervals for more than 1 hour, containing 37706 rows. Five mini-series exhibit a tendency towards '0' while two follow a Brownian motion. Anomaly detection uses all 7 time series in parallel, adopting the statistical approach by Tsay and applying the AD_M test proposed by Bontemps et al.

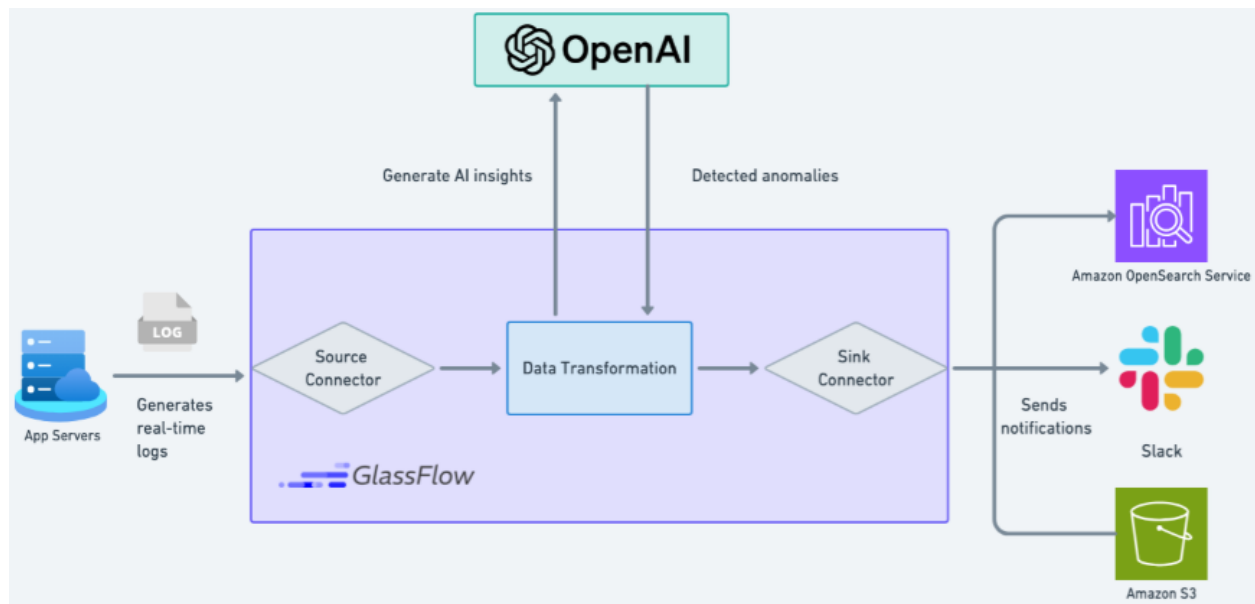
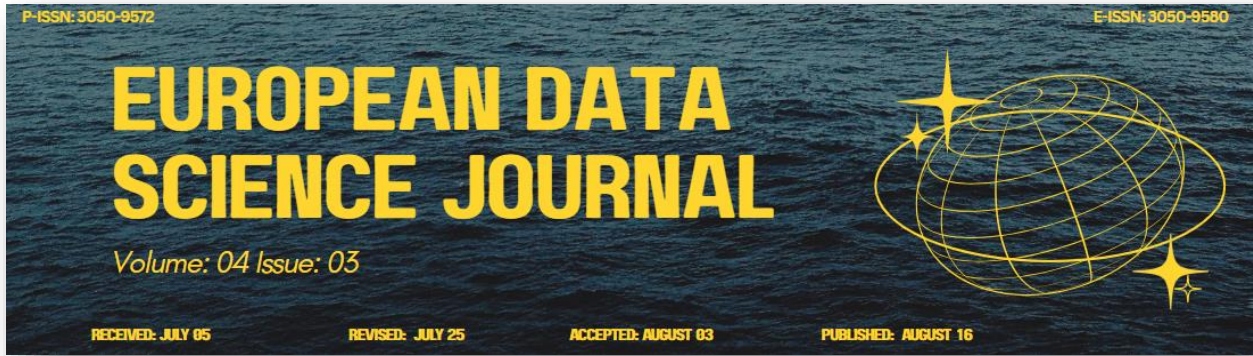


Fig 6: Real-time data anomaly detection and alerting

6.2. Dynamic pricing and inventory optimization

A streaming pipeline supports a second online business use case that combines time-series reinforcement learning for dynamic pricing and supplier order optimization on an e-commerce platform. A historical data feed continuously trains reinforcement learning agents with XGBoost. The agent for pricing recommends a new price per product-environment-geo-region pair every hour. The agent for inventory management decides whether and how much stock to order for all products in response to customer



demand in the next time period while considering the total inventory levels. When the pricing agent suggests a sale price lower than the production cost, the order-recommending agent refrains from ordering that product. The agents can overrule each other only on their respective decision variables.

The pricing agent offers sales deals by reducing the sale price of a random product at each recommending step. To enable a fair evaluation, the agent uses a random policy when the price remains unchanged (i.e., it has not been decided as a sale price). When the pricing agent suggests a product sale price higher than its production cost, the performance of the dynamic-pricing-and-inventory-management system improves, increasing the system-wide profit. The price-setting policy does not take into account the order-management agent's recommendation, and hence the results are different from those obtained when both agents are jointly optimized.

7. Evaluation and Best Practices

Real-time data pipelines are playing an increasingly key role in different domains of industry and services. Nevertheless, despite heavy investments, these systems do not reach the expected levels of reliability and efficiency. The development and execution of an experimental benchmark composed of five real-time data pipelines enables the identification of best practices for the design and implementation of Azure Databricks data engineering streaming workloads, based on the PySpark Streaming APIs. These pipelines address distinct integration challenges related to financial markets, e-commerce platforms, call-logging solutions, flood early-warning systems, and the automated decommissioning of unusual cloud resource usage. Several issues have emerged during the implementation and monitoring of these pipelines.

The findings point to potential optimizations of streaming jobs and resource allocation, the benefits of combining serverless Azure Functions with data lake and data warehouse solutions, and the impact of appropriate connectors on the overall latency of the process. Real-time data pipelines are essential to move fresh data to intelligent decision systems. Real-time data pipelines are essential to move fresh data to intelligent decision systems. The execution of a benchmark of Azure Databricks data engineering streaming workloads using the PySpark Streaming APIs enables the identification of best practices for the design and implementation of these systems.

Equation 5: Streaming anomaly score equation

$$z_t = \frac{x_t - \mu_t}{\sigma_t}$$

where:

- μ_t = rolling mean
- σ_t = rolling standard deviation

An anomaly decision can then be written as:

$$A_t = \begin{cases} 1, & |z_t| > \tau \\ 0, & |z_t| \leq \tau \end{cases}$$



Step-by-step derivation

1. We need to compare current observation x_t to “normal” behavior.
2. A standard measure of normal location is the rolling mean:

$$\mu_t$$

3. Deviation from normal is:

$$x_t - \mu_t$$

4. But raw deviation depends on scale, so normalize by rolling standard deviation σ_t :

$$z_t = \frac{x_t - \mu_t}{\sigma_t}$$

5. If the absolute standardized deviation exceeds threshold τ , label as anomaly:

$$|z_t| > \tau$$

6. Hence the binary anomaly rule:

$$A_t = \begin{cases} 1, & |z_t| > \tau \\ 0, & |z_t| \leq \tau \end{cases}$$

Interpretation

If $x_t = 145$, $\mu_t = 100$, $\sigma_t = 10$, then:

$$z_t = \frac{145 - 100}{10} = 4.5$$

If $\tau = 3$, then $4.5 > 3$, so:

$$A_t = 1$$

7.1. Benchmarking real-time pipelines

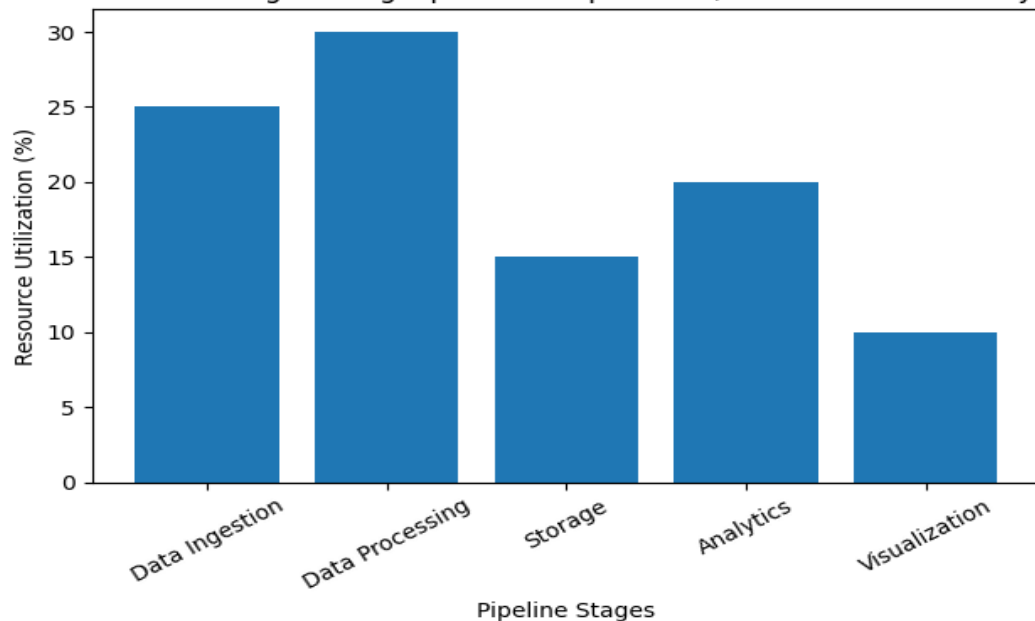
Micro-batching is the most common approach in streaming systems, but it increases latency and may not be sufficient for extreme time-critical use cases where data are ingested in real-time and the result must be computed and served in real-time, as



for the TV doesn't wait case. Furthermore, micro-batching is less efficient for low-throughput and/or low-latency workloads compared to native streaming processing (e.g. Firestore), and a mix of native streaming and micro-batching might achieve the best results. The standard benchmarking suite is Yahoo Streaming Benchmark, but it does not include detailed descriptions of the testbed or environment setup.

For sensitive workloads, reliability and consistency take precedence over performance, such as in isolation levels among transactions. For Delta Engine and Auto Loader, the performance improvement for schema evolution during write or auto schema discovery can be measured against other formats such as Apache AVRO with Delta Sharing. The interaction behavior of clients with the system under a specific workload can also be studied by analyzing the number of requests received per second and per client.

Real-Time Data Engineering Pipeline Components (Azure Databricks + PySp)



7.2. Performance tuning and resource optimization

The resource requirements and performance of multiple data engineering pipelines processing streaming data sources were benchmarked on Azure Databricks using the PySpark Structured Streaming APIs in micro-batch mode. Resource allocation estimates were generated by designing simulated end-to-end pipelines for the case studies using real data and query execution plans were captured to identify the most expensive features of the workloads. The results enabled the definition of best practices for tuning the performance and the resource utilization of streaming workloads implemented with PySpark.



Nine real-time data pipelines sharing a common architecture were created to process the New York City Taxi Trips dataset in three distinct ways. A first set of three pipelines ingested the original streaming source with original data integrity and without sampling; a second set leveraged cheating in the data and used validity masks to ignore cheating records; and a third set emulated a sampling stage in the data generation, ingesting only half of the original trips. These three groups of pipelines constituted a first set of experiments that operated in different data quality conditions; each experiment of the first set simulated three actual execution steps, and the total execution time was benchmarked.

8. Conclusion

Real-time data engineering pipelines support intelligent decision systems that, by construction and design, require up-to-date information for feature extraction, learning, and prediction. Real-time data pipelines based on streaming data technologies offer latest information for decision-making. This work studied aspects of real-time data engineering pipelines and provided a framework for practical implementations on Azure Databricks using PySpark.

Real-time data engineering pipelines exploit streaming data technologies to process data in a timely manner and support intelligent decision systems that require up-to-date information. Real-time data engineering pipelines were positioned in the context of intelligent decision systems and aligned with the concepts of streaming data and the architecture patterns by which real-time data engineering pipelines are designed. With the practical objective of demonstrating how real-time data engineering pipelines can be implemented to support intelligent decision systems, the focus was narrowed to the Azure Databricks platform and to the PySpark library for Python that is natively integrated into the platform. Two empirical use cases involving real-time data engineering pipelines operating in Azure Databricks were presented as demonstrations of recent work undertaken and as a starting point for future developments. The real-time data engineering pipelines were realized using the infrastructure and data sources made available in the Azure cloud environment and were built as micro-batches using the data as triggers architectural pattern. Actual cyclic patterns in real-time financial data provided a basis for benchmarking and supported dedicated performance evaluation of an anomaly detection and alert system for real-time market data.

8.1. Emerging Trends

The cloud-based Azure Databricks managed platform has emerged as a powerful tool for engineering data pipelines on micro-batch and real-time data streams. The Azure ecosystem provides a complete data engineering and analytics suite to integrate, query, visualize, and collaborate with real-time analytics and downstream machine learning data pipelines. The PySpark API set completes the Azure setup for developing intelligent decision systems on data in motion—real-time data engineering and analytics pipelines.

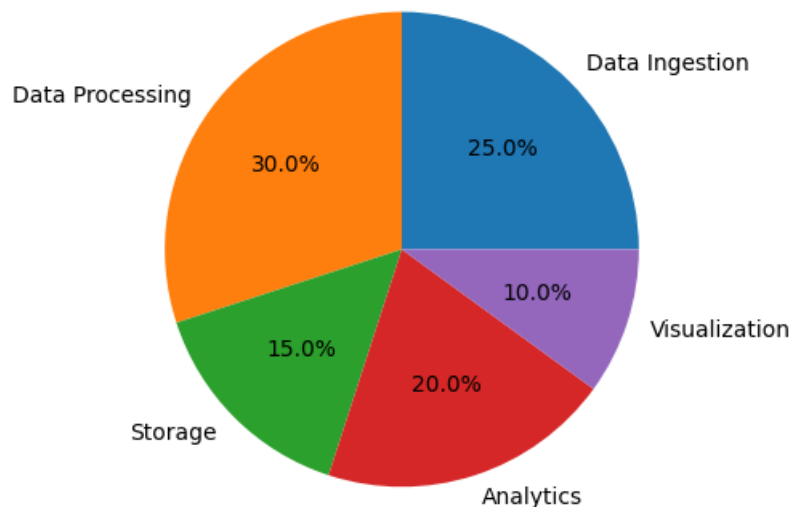
Two scripted applications demonstrate two typical decision-making areas of intelligent systems: real-time anomaly detection in transactional data and online pricing decisions in a cloud auto-scaling e-Commerce platform. Real-time streaming and online decision support data pipelines are key features knitted into the solution architecture using micro-batched Spark Streaming jobs for execution on Azure Databricks. Micro-batching input-stream sources and joining Connectors add custom constraints to keep resources on-demand and in control during execution.

Two scripted applications illustrate key decision-making domains in modern intelligent systems: real-time anomaly detection in transactional data and dynamic pricing in a cloud-based, auto-scaling e-commerce environment. These applications rely on real-



time streaming and online decision-support pipelines that are tightly integrated into the overall solution architecture. Leveraging micro-batched Spark Streaming jobs executed on Azure Databricks, the system ensures scalable and efficient processing of high-velocity data streams. The use of micro-batching for input sources, combined with customized connector-based joins, introduces deliberate constraints that help maintain resource efficiency, enabling on-demand scalability while keeping execution controlled and cost-effective.

Distribution of Pipeline Components



9. References

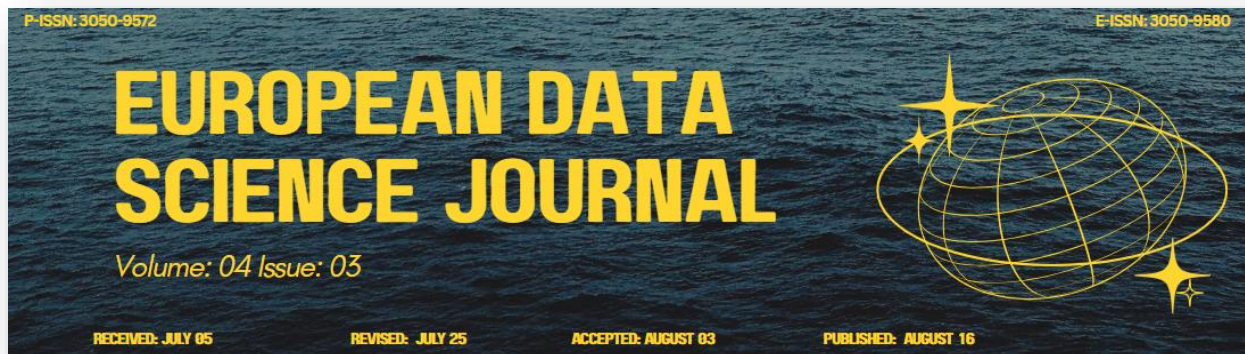
1. Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., Torres, J., van Hovell, H., Ionescu, A., Łuszczak, A., Świtakowski, M., Szafranski, M., Li, X., Ueshin, T., Mokhtar, M., Boncz, P., Ghodsi, A., Paranjpye, S., Senster, P., Xin, R., & Zaharia, M. (2020). Delta Lake: High-performance ACID table storage over cloud object stores. Proceedings of the VLDB Endowment, 13(12), 3411–3424.
2. Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. (2021). Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In Conference on Innovative Data Systems Research (CIDR 2021).



3. Yandamuri, U. S., Loganathan, R., Davuluri, P. S. L. N., Rani, P. R. S., Kolla, S. H., & Nagubandi, A. R. (2026). Adaptive Intelligence Networks for Humancentered Enterprise Automation and Governance. In 2026 Third International Conference on Innovations in Cybersecurity and Data Science (ICICDS) (pp. 678–683). IEEE. 2026 Third International Conference on Innovations in Cybersecurity and Data Science (ICICDS). <https://doi.org/10.1109/icicds70526.2026.11604640>
4. Akidau, T., Begoli, E., Chernyak, S., Hueske, F., Knight, K. E., Knowles, K., Mills, D., & Sotolongo, D. (2021). Watermarks in stream processing systems: Semantics and comparative analysis of Apache Flink and Google Cloud Dataflow. *Proceedings of the VLDB Endowment*, 14(12), 3135–3147.
5. Sheng, S., Huang, Q., Wang, S., & Bao, Y. (2021). PR-Sketch: Monitoring per-key aggregation of streaming data with nearly full accuracy. *Proceedings of the VLDB Endowment*, 14(10), 1783–1796.
6. Pandis, I. (2021). The evolution of Amazon Redshift. *Proceedings of the VLDB Endowment*, 14(12), 3162–3174.
7. Banoth, S., Santoshi Kumari, M., Lalitha, P., Deepthi, P., Kumar Peddi, R., & Kavitha, P. (2026). An Efficient Hybrid K-Means and Random Forest-Based Approach for Cloud Malware Detection and Privacy Protection. In 2026 6th International Conference on Intelligent Technologies (CONIT) (pp. 1–6). IEEE. 2026 6th International Conference on Intelligent Technologies (CONIT). <https://doi.org/10.1109/conit69683.2026.11621822>
8. Hai, R., Koutras, C., Quix, C., & Jarke, M. (2023). Data lakes: A survey of functions and systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(12), 12571–12590.
9. Khatiwada, A., Shraga, R., Gatterbauer, W., & Miller, R. J. (2022). Integrating data lake tables. *Proceedings of the VLDB Endowment*, 16(4), 932–945.
10. Behm, A., Palkar, S., Agarwal, U., Armstrong, T., Cashman, D., Dave, A., Greenstein, T., Hovsepian, S., Johnson, R., Krishnan, A. S., Leventis, P., Luszczak, A., Menon, P., Mokhtar, M., Pang, G., Paranjpye, S., Rahn, G., Samwel, B., van Bussel, T., van Hovell, H., Xue, M., Xin, R., & Zaharia, M. (2022). Photon: A fast query engine for lakehouse systems. In *Proceedings of the 2022 International Conference on Management of Data* (pp. 2045–2057). Association for Computing Machinery.
11. Armenatzoglou, N., Basu, S., Bhanoori, N., Cai, M., Chainani, N., Chinta, K., Govindaraju, V., Green, T. J., Gupta, M., Hillig, S., Hotinger, E., Leshinsky, Y., Liang, J., McCreedy, M., Nagel, F., Pandis, I., Parchas, P., Pathak, R., Polychroniou, O., Rahman, F., Saxena, G., Soundararajan, G., Subramanian, S., & Terry, D. (2022). Amazon Redshift re-invented. In *Proceedings of the 2022 International Conference on Management of Data* (pp. 2205–2217). Association for Computing Machinery.
12. Padma, G., Reddy, V. A. R., KA, S. D., Buvaneswari, B., & Kumar, K. S. (2026, April). Privacy-Preserved Face Recognition Biometric Authentication Using FaceNet and Zero-Knowledge Proofs for Secure, Access Control on Decentralized Blockchain Networks. In 2026 2nd International Conference on Intelligent Systems and Computational Networks (ICISCN) (pp. 1-8). IEEE.
13. Park, K., Saur, K., Banda, D., Sen, R., Interlandi, M., & Karanasos, K. (2022). End-to-end optimization of machine learning prediction queries. In *Proceedings of the 2022 International Conference on Management of Data* (pp. 587–601). Association for Computing Machinery.
14. Manousis, A., Cheng, Z., Ben Basat, R., Liu, Z., & Sekar, V. (2022). Enabling efficient and general subpopulation analytics in multidimensional data streams. *Proceedings of the VLDB Endowment*, 15(11), 3249–3262.
15. Sabek, I., Vaidya, K., Horn, D., Kipf, A., Mitzenmacher, M., & Kraska, T. (2023). Can learned models replace hash functions? *Proceedings of the VLDB Endowment*, 16(3), 532–545.
16. Sabek, I., & Kraska, T. (2023). The case for learned in-memory joins. *Proceedings of the VLDB Endowment*, 16(7), 1749–1762.
17. Mao, Y., Zhao, J., Zhang, S., Liu, H., & Markl, V. (2023). MorphStream: Adaptive scheduling for scalable transactional stream processing on multicores. *Proceedings of the ACM on Management of Data*, 1(1), Article 59.



18. Saxena, G., Rahman, M., Chainani, N., Lin, C., Caragea, G., Chowdhury, F., Marcus, R., Kraska, T., Pandis, I., & Narayanaswamy, B. (2023). Auto-WLM: Machine learning enhanced workload management in Amazon Redshift. In Companion of the 2023 International Conference on Management of Data. Association for Computing Machinery.
19. Chen, Z., Gu, Z., Cao, L., Fan, J., Madden, S., & Tang, N. (2023). Symphony: Towards natural language query answering over multi-modal data lakes. In Conference on Innovative Data Systems Research (CIDR 2023).
20. Kulkarni, V., Jyothsna, P., Musku, K., Mattaparthi, R., & Nagaraj, S. S. (2026, July). A Multi-Objective Metaheuristic Framework for Demand-Side Management in Isolated Tropical Photovoltaic Microgrids: Performance Bounds and Battery Trade-off Analysis. In 2026 14th International Conference on Smart Grid (icSmartGrid) (pp. 331-337). IEEE.
21. Sun, Y., Meehan, T., Schlusser, R., Xie, W., Basmanova, M., Erling, O., Rosa, A., Fan, S., Zhong, R., Thirupathi, A., Collooru, N., Wang, K., Agarwal, S., Gupta, A., Logothetis, D., Xirogiannopoulos, K., Dutta, A., Gajjala, V., Jain, R., ... Chattopadhyay, B. (2023). Presto: A decade of SQL analytics at Meta. Proceedings of the ACM on Management of Data, 1(2), 1–25.
22. Doshi, L., Zhuang, V., Jain, G., Marcus, R. C., Huang, H., Altinbükten, D., Brevdo, E., & Fraser, C. (2023). Kepler: Robust learning for parametric query optimization. Proceedings of the ACM on Management of Data, 1(1).
23. Verwiebe, J., Grulich, P. M., Traub, J., & Markl, V. (2023). Survey of window types for aggregation in stream processing systems. The VLDB Journal, 32(5), 985–1011.
24. Zhang, S., Soto, J., & Markl, V. (2024). A survey on transactional stream processing. The VLDB Journal, 33(2), 451–479.
25. Fragkoulis, M., Carbone, P., Kalavri, V., & Katsifodimos, A. (2024). A survey on the evolution of stream processing systems. The VLDB Journal, 33(2), 507–541.
26. Chatziliadis, X., Tziritza Zacharatos, E., Eracar, A., Zeuch, S., & Markl, V. (2024). Efficient placement of decomposable aggregation functions for stream processing over large geo-distributed topologies. Proceedings of the VLDB Endowment, 17(6), 1501–1514.
27. Schneider, J., Gröger, C., Lutsch, A., Schwarz, H., & Mitschang, B. (2024). The lakehouse: State of the art on concepts and technologies. SN Computer Science, 5, Article 449.
28. Gorle, R., & Pareek, P. K. (2026). Agentic AI-Based Security Orchestration for Oracle Cloud and Multi-Cloud Platforms. International Journal of Computer Information Systems and Industrial Management Applications, 18(11s), 354-369.
29. Levandoski, J., Casto, G., Deng, M., Desai, R., Edara, P., Hottelier, T., Hormati, A., Johnson, A., Johnson, J., Kurzyniec, D., McVeety, S., Ramanathan, P., Saxena, G., Shanmugam, V., & Volobuev, Y. (2024). BigLake: BigQuery’s evolution toward a multi-cloud lakehouse. In Proceedings of the 2024 International Conference on Management of Data (pp. 334–346). Association for Computing Machinery.
30. Xue, M., Bu, Y., Somani, A., Fan, W., Liu, Z., Chen, S., van Hovell, H., Samwel, B., Mokhtar, M., Korlapati, R. K., Lam, A., Ma, Y., Ercegovic, V., Li, J., Behm, A., Li, Y., Li, X., Krishnamurthy, S., Shukla, A., ... Zaharia, M. (2024). Adaptive and robust query execution for lakehouses at scale. Proceedings of the VLDB Endowment, 17(12), 3947–3959.
31. Okolnychyi, A., Sun, C., Tanimura, K., Spitzer, R., et al. (2024). Petabyte-scale row-level operations in data lakehouses. Proceedings of the VLDB Endowment, 17(12), 4159–4172.
32. Schulze, R., Schreiber, T., Yatsishin, I., Dahimene, R., & Milovidov, A. (2024). ClickHouse—Lightning fast analytics for everyone. Proceedings of the VLDB Endowment, 17(12), 3731–3744.
33. Mahadevan, S. (2026). Governed Serverless Automation Ecosystem for AI-Driven Enterprise Integration and Sustainable Cloud Operations. International Journal of Computer Information Systems and Industrial Management Applications, 18(16s), 1561-1570.
34. van Renen, A., Horn, D., Pfeil, P., Vaidya, K., Dong, W., Narayanaswamy, M., Liu, Z., Saxena, G., Kipf, A., & Kraska, T. (2024). Why TPC is not enough: An analysis of the Amazon Redshift fleet. Proceedings of the VLDB Endowment, 17(11), 3694–3706.



35. Harby, A. A., & Zulkernine, F. H. (2025). Data lakehouse: A survey and experimental study. *Information Systems*, 127, Article 102460.
36. Chandra, R., Chen, H., Matharu, R., Cai, S., Chen, J., Dutta, P., Ghita, B., Greenstein, T., Holla, G., Huang, P., Huo, Y., Ionescu, A., Ispas, A., Januschowski, T., Karajgaonkar, V., Leone, S., Lewis, D., Li, A., Li, N., ... Zaharia, M. (2025). Unity Catalog: Open and universal governance for the lakehouse and beyond. In *Companion of the 2025 International Conference on Management of Data* (pp. 310–322). Association for Computing Machinery.
37. Vakkalagadda, T., & Pareek, P. K. (2026). Self-Learning Artificial Intelligence Platforms for Autonomous Financial Risk Management. *International Journal of Special Education*, 41(15s), 238-254.