UtilitasMathematica

ISSN 0315-3681 Volume 122 (2), 2025

Real-Time Fraud ML on Spark Structured Streaming: Micro-Batch vs. Continuous Processing

Bhargav Vadgama
Staff Data Engineer, Austin, Texas, USA
Email: bvadgama2901@gmail.com

Abstract
Digital financial transactions have become highly vulnerable to fraudulent activities due to their growing volume and velocity, which necessitates real-time fraud detection systems. This article critically examines the analysis of machine learning-based fraud detection using Apache Spark Structured Streaming, which has two processing modes: Micro-Batch and Continuous Processing. A usable pipeline is trained and constructed on the IEEE-CIS Fraud Detection dataset, which incorporates feature engineering, supervised learning models, and stream processing to perform near-real-time fraud classification. Complex transformations and fault tolerance, supported by the Micro-Batch mode, have demonstrated reliability and analytical power with a marginal increase in latency. Continuous Processing mode offers significantly improved latency and throughput, is best suited for quickly issuing alerts in high-risk environments, and is limited in its transformations and recovery actions. Comprehensive experimentation compares the two modes in terms of latency, precision, recall, resource utilization, and reliability, assessing the operations that can be achieved. The results indicated that no one mode is inherently optimal; instead, they should be chosen according to particular use case guidelines. The paper concludes with practical advice, outlining the existing inadequacies of Spark used in Continuous mode, and presents opportunities for future exploration, including active learning, graph ML, and hybrid systems that would offer the best of both worlds in real-time fraud mitigation.

Keywords*: Real-Time Fraud Detection, Apache Spark Structured Streaming, Micro-Batch vs. Continuous Processing, Machine Learning Inference, Scalability and Latency*

1. Introduction

The surging digital financial interaction in e-commerce, mobile banking, and fintech platforms has brought immense threats and opportunities. Fraudulent activities have, however, been evolving both in sophistication and rate with the increased volume and velocity of financial data, which has been soaring so far. The historical data analysis-based and rule-based traditional fraud detection systems are no longer adequate. These systems are known to operate in batches, and in most cases, fraud is detected after transactions have been processed. By that time, it may already make a significant difference, particularly in high-velocity zones, such as credit card approvals, instant payments, or cryptocurrency transactions.

Real-time fraud detection has become an essential need in various industries that involve monetary or digital transactions, including finance. Real-time systems, unlike retrospective analysis, observe transaction streams in real-time, detect anomalies or patterns suggestive of fraud, and alert or respond in milliseconds to seconds. The architecture must enable low-latency processing, horizontal scalability, and support for machine learning-based decision-making to meet such time-sensitive needs. That has created increased interest in stream processing systems that can process high-throughput data streams in real-

time settings. Apache Spark Structured Streaming is one of the most powerful technologies available today for processing real-time data. Structured Streaming is built on top of the popular Apache Spark engine, providing a unified platform for processing both batch and streaming data using a compatible DataFrame API. It offers fault-tolerant processing, windowed computations, complex event processing, and integration with the most popular tools like Apache Kafka, AWS Kinesis, and JDBC-compatible sinks.

Structured Streaming is primarily available in two modes: Micro-Batch Processing and Continuous Processing. In Micro-Batch mode, the system reads incoming data in small time-based batches (e.g., every 2 seconds). Such a mode provides a high degree of consistency and throughput capacity, making it applicable to all Spark operations, including joins, aggregations, and stateful transformations. It is appropriate in complex analytic operations that can tolerate a modest amount of latency. Conversely, the Continuous Processing mode receives events one at a time and can provide latency as low as 150 milliseconds. This is a record-by-record model of execution designed for use with applications that require an instantaneous response. However, it does not cover all Spark operations and is as yet under development in most production setups. As machine learning (ML) inference becomes increasingly real-time, fraud detection models deployed within such streaming architectures present new possibilities and considerations. Micro-batch versus continuous processing not only influences the responsiveness of a system but also affects the performance of the model, the efficiency of resources, and the difficulty of feature engineering in streaming problems.

This paper discusses the use and testing of real-time fraud detection with machine learning in the Apache Spark Structured Streaming platform. Its goal is to design an A-to-Z pipeline that can ingest data on transactions, run feature engineering, utilize an ML model to score fraud, and present outcomes in near real-time. The main idea will be to compare the Micro-Batch and Continuous Processing regimes in terms of pragmatic performance, focusing on differences in latency, throughput, model performance, and scalability. Through the evaluation of transactional data and a production-like Spark environment, the analysis will enable system architects, data engineers, and fraud analysts to select the most suitable processing strategy for specific fraud detection cases. The discussion also outlines actionable suggestions and constraints that may characterize each strategy and presents insights into how machine learning and real-time stream processing can be effectively integrated to neutralize financial fraud in the current diverse online world.

2. Understanding Real-Time Fraud Detection

2.1 Definition and Characteristics

The process of detecting potentially fraudulent activity within milliseconds of a transaction or user action is real-time fraud detection (3). Real-time systems differ from conventional fraud systems in that they operate based on historical data and make assessments based on it later. In contrast, real-time systems make assessments instantaneously, detecting threats in real-time. This is required in contemporary financial settings, where online transactions are authorized and completed within a near-instant period.

The responsiveness to the current situation is the main feature of real-time detection. These are lines of upcoming events, such as a payment request, a login, and an account change, that routinely involve making a judgment against a list of dynamic rules or predictive models without disrupting the user experience. These systems frequently include event-based architectures that continuously track data flows

and raise instant alerts (by flagging transactions, aborting processing, or sending alerts to fraud units) in under one-thousandth of a second. The prioritization of immediacy renders constant observation, speed of inference, as well as low-latency software, particularly instrumental in any instantly alerting pattern of fraud detection.

As illustrated in Figure 1, such systems rely on low-latency software and rapid inference mechanisms to ensure minimal delay in fraud detection, highlighting the importance of continuous monitoring and real-time decision-making in safeguarding digital financial interactions.

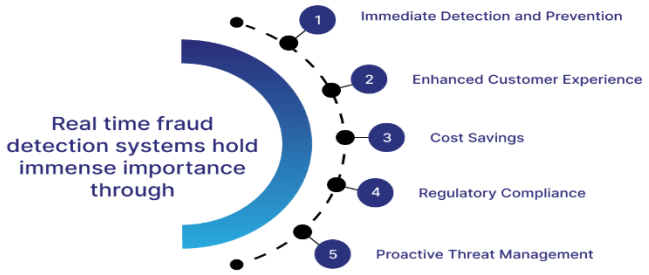

Figure 1: Understanding Real-Time Payments Fraud Detection: Everything You Need for Fortifying Payment Security

2.2 Financial Fraud Types

There are a few types of fraud prevalent in online financial systems (28). Transaction fraud is one of the most widespread, as unauthorized users can use stolen credentials or hacked cards to purchase or withdraw funds. This type of fraud is typically detected through the analysis of uncertain transaction patterns, such as irregular amounts, unfamiliar terminals, or transactions originating from unusual geographical regions (9,10). To support such real-time detection, scalable and consistent data systems like MongoDB are increasingly adopted, as they offer high performance while effectively managing large volumes of transactional data.

Account takeover constitutes yet another major threat that occurs when an unauthorized user gains access to a genuine user's account. Changes in the password and the manner of using the device, as well as the sudden transfer of funds after authorization using an IP address unknown to the user, can be signs of such conduct. CNP fraud can be found in online retail and mobile commerce channels, which do not require the presence of a physical card to make a purchase. Since such transactions cannot involve physical mechanisms of verification, such as chip entry or PIN, they are an attractive vehicle for fraudsters with stolen card data. This can be countered by real-time systems examining metadata, including device fingerprints, browser behavior, and IP reputation. A larger and more complex type is synthetic identity fraud, in which the attackers piece together real and fake identities to form entirely new ones. These false identities tend to undergo preliminary credit inquiries successfully and can be cultivated over time to facilitate larger fraud schemes. To identify the synthetics in real-time, detecting advanced profiling and behavioral analytics cannot be provided merely by transaction checking.

As shown in Table 1, these fraud categories—such as transaction fraud, account takeovers, and synthetic identity fraud—display specific common indicators that can be monitored in real-time to trigger alerts or initiate further verification processes.

Table 1: ***Common Types of Financial Fraud in Online Systems***

Fraud Type	Description	Common Indicators
Transaction Fraud	Unauthorized use of stolen cards or credentials to perform transactions	Unusual amounts, unfamiliar devices, strange geolocations
Account Takeover	Illicit access to a user's account by an attacker	Sudden password changes, new devices, IP mismatches, and quick fund transfers
Card-Not-Present (CNP)	Fraud in online/mobile transactions without physical card presence	No chip/PIN use, suspicious metadata (device, browser, IP reputation)
Synthetic Identity Fraud	Creation of fake identities using real and fabricated data	Passed credit checks, long inactive periods followed by large transactions

2.3 Major Hurdles in Real-Time Detection

Implementing live fraud detection systems presents several technical and analytical challenges. One of the most critical challenges is the volume and velocity of data. Financial institutions and payment gateways process thousands of transactions every second (21). Each transaction must be ingested, processed, and scored in real-time—often within 100 milliseconds or less—to avoid delays or disruptions to the customer experience (17). This demand places immense pressure on the underlying infrastructure to ensure both security and performance under high loads. Another problem is the dataset's imbalanced nature. In the majority of cases, less than 1 percent of transactions are related to fraud. This imbalance in the classes may lead to difficulties in training machine learning models, as they may not be able to learn to generalize sufficiently to recognize rare anomalies without an excessive number of false positives. The issue with poorly balanced data is that it often requires specialized data management methods, such as oversampling, undersampling, or cost-sensitive learning strategies.

There is another complexity of concept drift. There is constant change in the pattern of fraud as fraudsters adapt to detect the systems that have been set up. The methods, which were helpful two months ago, may become inapplicable, and the models will degrade over time. To remain up-to-date, real-time systems should thus be capable of incorporating adaptive learning schemes or retraining the model regularly. Lastly, a very low latency in fraud prediction imposes pressure on infrastructure, model design, and feature engineering. The fraud detection logic, even for complex fraud, needs to be streamlined so that it can run in under a second. This constrains the deployment of algorithmically intensive work and promotes the adoption of simplified models that incorporate both the predictive power and performance benefits. A combination of these aspects underscores the need for an efficient, scalable, and flexible architecture that can not only detect real-time fraud but also quickly adapt to new threats without compromising the system's speed or user trust.

As illustrated in Figure 2, these interrelated challenges underscore the necessity for fraud detection architectures that are not only scalable and efficient but also flexible enough to adapt to changing fraud patterns without compromising system performance or user trust.

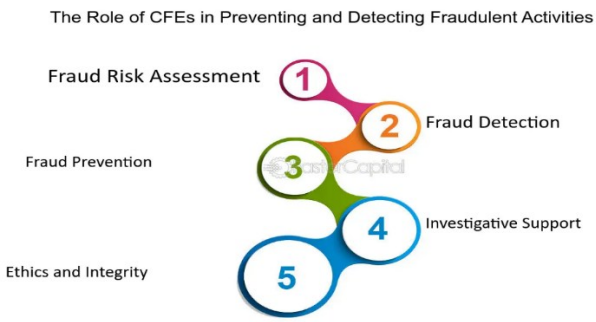

Figure 2: ***Preventing and Detecting Financial Fraud***

3. Spark Structured Streaming: An Overview

3.1 What is Spark Structured Streaming?

Spark Structured Streaming is a fault-tolerant and distributed stream processing engine implemented on top of the powerful SQL engine of Apache Spark (25). It enables programmers to develop real-time data processing systems through familiar Spark SQL APIs and hides much of the complexity that streaming systems have traditionally entailed. The advantage of Structured Streaming is that it treats streaming data as an unbounded table, where new data is continuously processed in a declarative and consistent incremental fashion. The united architecture it provides is one of its significant advantages. Streaming also uses the same DataFrame or Dataset APIs as those used for batch processing, allowing easier writing, testing, and maintenance of real-time pipelines without the need to learn new programming paradigms. Such a method facilitates the development of applications that require both historical data analysis and real-time stream processing within the same pipeline.

3.2 Sources of Stream Ingestion

Structured Streaming is compatible with a variety of data sources that are typical of real-time systems. Apache Kafka is one of the most popular open-source systems, designed to provide a high-throughput, distributed messaging system that suits real-time event streams, such as transaction log history or user activities. Spark can connect to Kafka topics and read these topics exactly once. Another source supported is Amazon Kinesis, which is especially helpful to organizations that use AWS infrastructure. It offers a safe, cloud-native deployment of continuous delivery of real-time data records from multiple producers. It can also be file-based streaming. Spark supports files as streaming inputs, whereas Spark can monitor the directories of HDFS or Amazon S3. It helps systems that record event logs as batch files with time intervals. Furthermore, Structured Streaming can consume data via network sockets or user-defined data sources through structured APIs, allowing for integration with non-standard or proprietary systems.

3.3 Essential Streaming Ideas

Structured Streaming offers fundamental concepts to enable efficient and accurate real-time data processing (22). One such important distinction is between *event time* and *processing time*. Event time refers to the actual time at which an event occurs, whereas processing time denotes the time when Spark ingests and processes the event. Understanding this distinction is crucial for accurately sequencing event-time records and effectively handling out-of-order data (29, 30). These timing concepts are foundational in building scalable and reliable streaming systems, particularly in domains such as healthcare communication and outcome monitoring, where timing precision directly impacts system responsiveness and user experience. To process late-coming data, watermarks are employed by declaring a threshold of

tolerable lateness. This allows for avoiding uncontrolled state extension in aggregations and proper results in time-window operations. Aggregations, joins, and deduplication are examples of stateful operations that require an intermediate state to be maintained between events. Structured Streaming manages states ourselves, which allows complex real-time analytics with deterministic results.

4. Micro-Batch vs. Continuous Processing Modes

4.1 Micro-Batch Mode

The Spark Structured streaming implementation principle is micro-batch processing. In this mode, the received stream is chopped into small batches within a specified interval, such as every one or two seconds. The micro-batches are considered individual Spark jobs and are processed accordingly. Although this design incurs a modest latency cost, it provides unrestricted access to Spark's rich feature sets, including windowed aggregations, stateful joins, and complex data transformations. Stability and maturity are among the key benefits of micro-batching. It is widely used in production system applications and can maintain overall fault tolerance through checkpointing. In the event of a node or job failure during processing, Spark allows the replay of the data from the last successful batch, ensuring it is not lost or duplicated.

This model is an efficient compromise of latency and functionality in fraud detection applications. Most fraud cases involve a history, such as the amount of the last ten logins, the average transaction value in each region, and last-hour device switching, among other factors. These patterns require stateful and time-based aggregations, all of which can be done in a micro-batch model. Furthermore, Spark will enable machine learning models used in production through micro-batch pipelines to reap the benefits of Spark broadcasting. This mechanism allows models to be delivered to workers consistently, ensuring consistency during scoring within each batch cycle. Although introducing this form of minimal latency (generally between 300 and 700 milliseconds), micro-batch processing remains beneficial when applied to systems that prioritize analytic complexity, reliability, and elastic data enrichment over sub-second response rates.

As illustrated in Figure 3, micro-batch processing in Spark divides the continuous data stream into manageable processing units, enabling both high-throughput computation and support for rich streaming analytics.

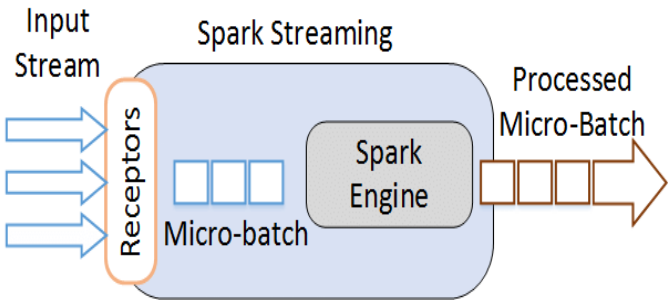

Figure 3: Micro-batch processing used in Spark stream.

4.2 Continuous Processing Mode

Continuous Processing mode offers a distinct approach, as it executes actions for each input record independently (5). The system does not use batch intervals; instead, it processes events as they are received, allowing end-to-end latencies of 20 to 80 milliseconds. This ultra-low-latency model is ideal for time-bound applications, where time is money. For example, transaction monitoring of large fund transfers

or real-time fraud alerts for key systems fall into this category. However, with this increased responsiveness, there are limitations. At this point, only a focused subset of stateless transformations can be used in continuous mode. Performance: Operations that are annotated as involving long-running states, e.g., aggregates, joins, or event deduplication, are not yet supported. Through this, use cases based on this model will need to utilize less complex, lightweight logic that can be executed individually on each occasion.

Another reflection is that the continuous mode remains in active development and, when used, is still essentially experimental in most Spark distributions. Specific operational characteristics, including built-in recovery capabilities, watermarks, and metric reporting, are either compromised or nonexistent. This can be a problem where a very high degree of reliability or complex observability of failures is needed. For real-time fraud detection, continuous processing can be effectively utilized to perform prime-time filtering or scoring, where the detection logic is straightforward and action must be taken in real-time. These could be suspicious amounts of money, blocked account payments, or high-risk geolocation detection in real-time.

4.3 Comparison Summary

There are various trade-offs to consider when choosing between the two modes. Micro-batch mode is more flexible, tolerant to faults, and transformative, making it suitable for most real-time ML pipelines. Continuous processing is weak in terms of its support for the feature set, but scales well when latency is paramount and the detection logic can remain stateless.

As shown in Table 2, micro-batch processing provides greater flexibility, full transformation support, and enhanced stability, making it suitable for stateful analytics and complex fraud modeling.

Table 2: Comparison between the two modes reveals key differences

Metric	Micro-Batch	Continuous
Latency	300–700 ms	20–80 ms
Flexibility	High	Low
Transformation Support	Full	Limited
Stability	High	Experimental
Use Cases	Stateful analytics	Real-time alerts

Selecting the appropriate mode depends heavily on the operational context. In cases where complex pattern recognition, feature aggregation, and reliability are essential, micro-batch mode remains the preferred choice. Conversely, for high-priority alerts that require immediate response without heavy computation, continuous mode offers a compelling solution despite its current limitations.

5. Machine Learning for Fraud Detection

5.1 ML Model Selection

Machine learning is a notable feature that enables real-time fraud detection, particularly through supervised models designed to distinguish between legitimate and fraudulent transactions. Commonly used algorithms include logistic regression, decision trees, random forests, and gradient-boosted techniques like XGBoost. These models differ in terms of complexity, training time, and interpretability (7,

8). Implementing such models within fault-tolerant, event-driven systems is critical for ensuring consistent performance and reliability, especially in dynamic and high-volume financial environments. Logistic regression provides an easy and helpful backup model. Its linear decision boundary leads to efficiency during scoring in streaming scenarios and provides transparency around how the features affect predictions. Nevertheless, it might perform poorly in situations where there are non-linearities in relationships in the data.

Decision trees, however, can express more complicated relationships between variables at the expense of being prone to overfitting unless regularized or pruned (23). Random forests are a possible solution to this weakness, utilizing a collection of decision trees through the technique of bagging, which involves a tradeoff that enhances generalization but incurs a computational cost. XGBoost is also recognized for its robust predictive performance and ability to handle structured data. It employs gradient boosting, which helps refine weak learners iteratively and tends to surpass simple models in fraud detection tasks. However, being more complicated implies a longer inference time, which can be important when considering its use in low-latency streaming applications. Ensemble models tend to exhibit improved performance when predictions from multiple learners are combined. However, they are generally more complex, require additional resources, and must be used with caution in real-time contexts.

5.2 Feature Engineering Techniques

The quality and relevance of features supplied during training and inference are the most significant factor that determines the quality of a fraud detection model. In real-time, features need to be engineered to be both predictive and efficient. Among the key categories are geolocation-type features, such as the detection of anomalies in IP addresses compared to the historical locations of a specific user's transactions. Sudden geolocation changes may be powerful signs of fraud. Context is further augmented by behavioral profiling, such as comparing the value of a transaction with the user's typical spending patterns. As another example, an extensive money transfer process initiated by a user deemed low risk may be treated as benign, but an account with a history of low spending is likely to be treated as suspicious.

Time-based features also play a vital role. They indicate the number of transactions that took place in particular periods, the number of logins attempted, or the sharpness of an increase in actions on a specific device or IP. The characteristics can identify patterns of fraud, such as credential stuffing or transaction bursts. By correlating transactions with known or unknown devices through device fingerprinting, which utilizes unique identifiers such as browser version, screen resolution, or OS, the accuracy of the model can also be improved.

5.3 Imbalanced data

The detection of fraud is a problem of imbalanced classification in itself since the number of genuine transactions greatly exceeds the number of fraudulent transactions (20). This over-fitting may lead to bias in models to the majority class, which is seen in high overall accuracy but low recall on the minority (fraudulent) class. This problem is addressed through several strategies. A simple yet effective approach is to oversample the minority or undersample the majority classes. Nevertheless, oversampling can lead to overfitting, particularly in cases where fraudulent samples are duplicated without any variance. Such methods as SMOTE (Synthetic Minority Oversampling Technique) contribute to its suppression by creating synthetic records of the minority classification, relying on the similarity of feature spaces. It can enhance the model and improve performance without incurring the disadvantages of simple duplication. In practice, in a production setting, the model can also prioritize the detection of rare fraud events, adjusting class thresholds or utilizing cost-sensitive learning, where misclassifying fraud is more expensive.

As illustrated in Figure 4, the distribution imbalance between classes and strategies such as SMOTE is a critical consideration in the design of robust, production-ready fraud detection systems.

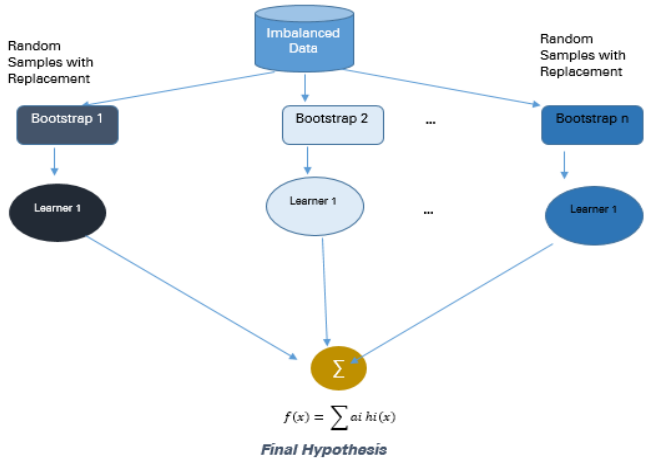

Figure 4: imbalanced-data-classification

5.4 Streaming Inference

When deploying a trained model into a live Spark™ Structured Streaming pipeline, key factors such as latency, consistency, and scalability must be carefully addressed (35). A common approach is to broadcast the trained model across all Spark worker nodes, thereby avoiding the computational overhead of reloading the model for each batch or event. This ensures that all nodes use the same model version during inference, maintaining consistency across the distributed system (26). Efficient model distribution strategies, such as this one, are essential for minimizing inference delays and maximizing system throughput. The model's prediction function can be called by a scoring mechanism, such as User-Defined Functions (UDF). This enables flexibility but, at the same time, creates performance overhead. Execution can be made faster by using pandas UDFs or MLlib native models.

Another factor is the update rate of models. Since there is a concept drift in the fraud patterns, models must be retrained and redeployed frequently. Although Spark does not intrinsically support online learning, third-party scheduling, and model management frameworks can be utilized to periodically refresh the models and send them into the pipeline without interruption. The combination of these considerations means that machine learning models can not only have high accuracy in fraud detection but also be efficient enough to work within the limits of a real-time streaming architecture.

As shown in Table 3, two commonly used model deployment methods are broadcasting the model to all workers and loading the model per batch or event. The broadcasting method is highly efficient for production pipelines, ensuring that each worker node uses a consistent model copy during scoring. However, it may require a manual strategy to update models during runtime.

Table 3: ***Model Deployment Strategies in Spark Structured Streaming***

Deployment Method	Description	Pros	Cons
Broadcast Model	Distributes trained model to all Spark workers	Low latency, consistent version	Requires manual update strategy

Deployment Method	Description	Pros	Cons
Load per Batch/Event	Loads model during every batch or record	Easy to implement	High computation cost, slow

6. System Architecture and Data Pipeline

6.1 High-Level Architecture

A high-performance system for detecting fraud in real-time relies on a properly orchestrated system with high throughput, low latency, and the ability to incorporate streaming data and perform machine learning inference easily. Its fundamental architecture employs modular and scalable patterns, including Kafka, Spark, Structured Streaming, and ML Model Sink (PostgreSQL, Elasticsearch, Kafka). Apache Kafka in this pipeline serves as the real-time ingestion layer for transactions. Apache Spark Structured Streaming serves as the processing engine, taking data consumed through Kafka, transforming features, executing machine learning models, and returning the results. The ultimate predictions are sent to an output sink, which can be a Sink (e.g., PostgreSQL, Elastic search, or another Kafka topic), depending on how the alerts or decisions should be consumed. Such an architecture also guarantees the decoupling of specific portions, allowing components to be more dynamic by being comfortably scaled or replaced with other components without necessarily disrupting the entire pipeline. As shown in the example, different models can be deployed without requiring changes to the ingestion layer or the introduction of new visualization tools downstream.

As illustrated in Figure 5, the design emphasizes modularity, real-time responsiveness, and the ability to maintain low-latency inference pipelines while supporting high-frequency financial transactions.

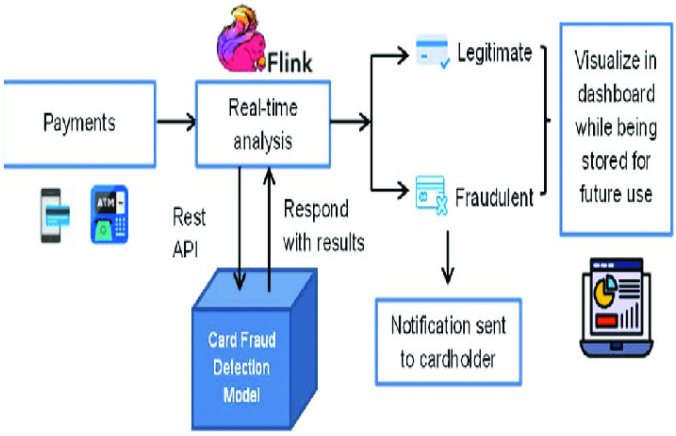

Figure 5: Live fraud detection architecture

6.2 Components Analysis

The first essential component is the Kafka producer, which functions as a simulator or a live source of financial transaction data within the system (27). In real-world deployments, producers can include transaction gateways, mobile applications, banking APIs, or payment processors. These producers typically emit data in JSON or Avro formats in real-time, which is then streamed into Kafka topics. Each message generally carries metadata such as transaction ID, user ID, amount, currency, device ID, geolocation coordinates, timestamp, and other relevant details (18). Leveraging such structured metadata supports downstream analytics, including fraud detection and business intelligence workflows.

The Spark Structured streaming job subscribes to the Kafka topic and reads the data with the Kafka connector in Spark (11). After consumption, the data undergoes transformation processes, including parsing, feature extraction, enrichment with external data (e.g., geolocation look-up or device profiling), and time alignment. Spark utilizes this pre-trained machine learning model to analyze each transaction and calculate the probability of fraud. The scoring step will be executed either with a UDF representing custom model logic or an in-built Spark MLlib model, depending on the model format and pipeline construction.

To maintain a steady performance, Spark jobs get built using adequate checkpointing, parallelism, and resource management. Checkpoint directories maintain the state in micro-batch mode, enabling fault-tolerant resumption in the event of job failure or node unavailability. The final predictions, which include the probability of fraud, a decision flag (indicating whether the transaction is fraudulent or not), and a confidence score, are output to a sink. The PostgreSQL database is typically applicable where a structured storage environment and downstream queries are desired. In the case of visual analytics and dashboards, Elasticsearch provides real-time indexing and searching capabilities. Alternatively, fraud flags may be posted back to Kafka topics to be consumed by alerting systems, fraud operations dashboards, or auto-response services. This architecture enables real-time analytics, affirms monitoring of the operation, and automated mitigation efforts. A unification of powerful stream processing, precise model inference, and flexible output channels provides the foundation for a scalable fraud detection system ready for production and capable of addressing new threat patterns.

7. Methodology

7.1 Dataset Preparation

To determine the effectiveness of real-time fraud detection by applying a machine learning algorithm in a Spark Structured Streaming, the IEEE-CIS Fraud Detection dataset is prepped. The dataset is widely published for academic and industry benchmarking purposes, consisting of transactional records labeled with fraud indicators that feature both numerical and categorical features relevant to behavioral analysis. Since the first dataset follows a static approach intended for batch processing, it must be transformed into a streaming format to enable real-time analytics. The records are converted into JSON or Avro messages to simulate live transaction events. These messages are then published to Apache Kafka topics, with event timestamps preserved to mimic the flow of actual financial transactions. This transformation supports event-driven processing, allowing the streaming system to handle data more efficiently and realistically, closely reflecting production environments (24). Such an approach improves the system's responsiveness, adaptability, and alignment with real-world operational needs.

7.2 Stream Feature Extraction

The pipeline cannot be complete without regard to real-time feature engineering, which directly affects model performance and predictive accuracy. Basic features, as well as the derived ones, are computed in real-time by the streaming system. Time-based aggregations are used to calculate window-based features, such as the total number of transactions made within the last five minutes or the unique IP addresses a user has used during a specific period. These characteristics will enable the detection of any anomalies and unforeseen deviations in behavioral patterns, suggesting a potential case of fraud. Also, the real-time stateful metrics support the recording of past information. An example is that Spark maintains running averages of the transaction amounts per user, allowing each new event to be compared against a learned behavioral baseline. This form of continuous computing of contextual features, performed within

a micro-batch, is supported by Spark, which is crucial for models that rely on practical pattern recognition rather than fixed attributes.

As illustrated in Figure 6, the feature engineering pipeline dynamically transforms raw event streams into enriched, behaviorally contextualized features used for model scoring.

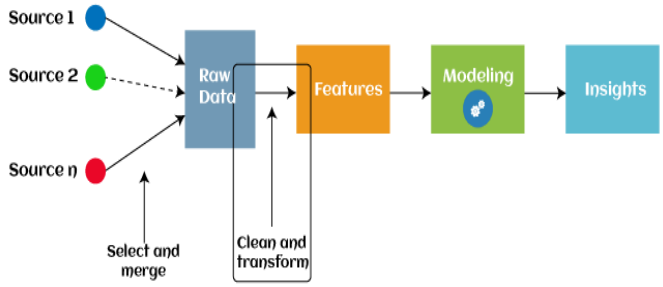

Figure 6: feature-engineering-pipeline

7.3 Trained ML Model Deployment

The offline training of the machine learning model uses labeled and prior data from the IEEE-CIS dataset. Preprocessing and feature engineering follow, and several models are tested on their performance. There is a trade-off between the precision of prediction and the speed of the model implementation, and the Random Forest-based model or XGBoost-based model can be applied because of their effectiveness in discriminating between a legitimate and a fraudulent. The trained model is next serialized and inserted into the Spark pipeline in either of two ways. The former employs PySpark User-Defined Functions (UDFs), in which the scoring procedure of the model is contained within the stream processing logic. The second method is to export the model as Predictive Model Markup Language (PMML), thereby enabling the model to be used in a language-independent and consistent manner within the cluster.

7.4 Configuration of Modes Streaming

The streaming application is available in two versions, allowing for comparison of operations with various processing modes (14). To adjust the micro-batch type, set the size of the trigger interval to be after every two seconds by applying the trigger (processing Time = two seconds). This is a type of setup that uses small batches of records periodically and can constantly transform entirely, and also use complicated aggregations. Continuous processing is set up with a trigger (Continuous ("1 second")) so that the system performs each incoming record as a single processing unit. This setup is applied to test low-latency behavior at the expense of lesser support for transformation.

7.5 Measures of Evaluation

The assessment of fraud detection systems relies on a range of general metrics, including performance, the efficacy of machine learning, and the responsiveness of the system. The metrics include average and maximum latency, throughput in records per second, and usage of system resources, such as CPU and memory. These indicators offer insight into the scalability and responsiveness of each processing mode. The accuracy, precision, recall, F1-score, and ROC AUC are machine learning indicators that quantify how well the model distinguishes between fraudulent and legitimate transactions. These measurements are calculated in real-time within the pipeline to evaluate the quality of the models. Operational metrics play a crucial role in assessing the effectiveness of fraud detection systems. One key metric is alert response time, which refers to the duration between the occurrence of a suspicious transaction and the issuance of a corresponding fraud alert. Another critical measure is the false positive rate (FPR), which tracks the

frequency of legitimate transactions incorrectly flagged as fraudulent. A high false positive rate (FPR) can erode customer trust and impose unnecessary workload and costs on fraud response teams (32,33). Monitoring and optimizing these metrics is essential to ensure timely and accurate fraud mitigation in real-time financial systems. Collectively, these measurements provide a comprehensive assessment of the micro-batch and continuous setups with typical streaming ML workloads. This approach implies that the comparison will capture both the technical implementation and the practical detection results of fraud.

As shown in Table 4, evaluation metrics are typically categorized into system performance, machine learning (ML) model performance, and operational metrics.

Table 4: Evaluation Metrics for Real-Time Fraud Detection Systems

Category	Metric	Description
System Performance	Average Latency	Average time taken to process each record
	Maximum Latency	Longest time taken for any single record to be processed
	Throughput	Number of records processed per second
	CPU & Memory Usage	Resource utilization indicating system load and scalability
ML Model Performance	Accuracy	Proportion of total correct predictions
	Precision	Fraction of flagged transactions that are truly fraudulent
	Recall	Fraction of actual fraud cases that are correctly flagged
	F1-Score	Harmonic mean of precision and recall
	ROC AUC	Measures model's ability to distinguish between fraud and legitimate cases
Operational Metrics	Alert Response Time	Time between a suspicious transaction and fraud alert being issued
	False Positive Rate (FPR)	Percentage of legitimate transactions wrongly flagged as fraud

8. Implementation and Experimentation

8.1 Tools and Technology Stack

A real-time fraud detection system stack must be robust, low-latency, and highly scalable to efficiently handle continuous data streams. It should support high-throughput data storage and processing while seamlessly integrating machine learning at all layers. At the core of this architecture lies Apache Spark 3.x with its Structured Streaming engine, which enables the construction of real-time data pipelines. Spark provides a high-level API for defining streaming queries and applying complex transformations to live data. Its built-in fault tolerance and horizontal scalability make it suitable for production-grade fraud detection environments (34). This foundation ensures reliable performance even under demanding workloads. Apache Kafka is used as the messaging backplane, emulating real-time transaction streams by publishing events into topics that are consumed by Spark jobs. Kafka is durable, partitionable, and provides high throughput, thus offering a consistent ingestion layer in the fraud detection pipeline (4).

The pipeline logic can be written in either PySpark for Python-based ML model integration or Scala for performance-optimized streaming applications. PySpark is best suited when a team already has a set of Python ML tools, as it enables the simple serialization and scoring of models through Python User-Defined Functions (UDFs). Concerning containerization and orchestration, Docker will be applied to bundle specific services. e. Kafka brokers, Spark jobs, and output sinks into individual containers. Kubernetes can also be used optionally to manage scale in deployments, especially in cases where managing and hosting multiple parallel Spark applications with dynamic resource allocation is required.

As illustrated in Figure 7, this stack forms an intelligent, flexible, and modular architecture that supports scalable, real-time fraud detection in various environments, including insurance, banking, and e-commerce.

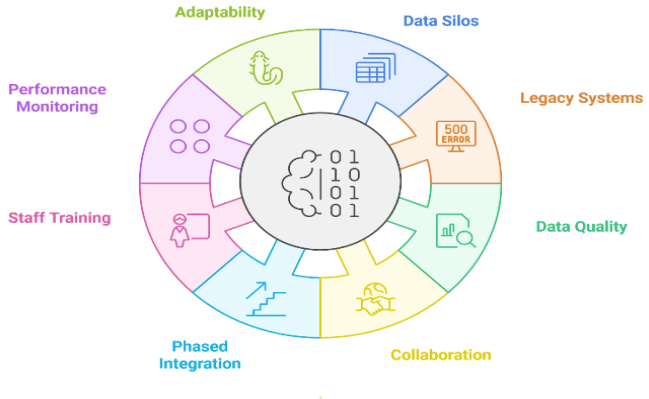

Figure 7: AI-based-insurance-fraud-detection

8.2 Experiment Preparation

To achieve a resemblance to real-life scenarios, a Kafka producer stream of 500 to 1000 transactions per second is emulated in the experiment. Transaction metadata, including user ID, amount, timestamp, device ID, and geolocation, is present in each message. The streaming pipeline is implemented on a dedicated machine with 4 CPU cores, 16 GB of memory, and storage with SSD to reduce the I/O latency. This is a medium-level hardware profile that is expected to be sufficient for development work, as well as early-level production testing. The implementation is performed in one of two ways: through interactive notebooks for experimentation and debugging or through production-ready scripts on clusters to specify Spark jobs, resource configurations, and Kafka topics. With this dual-mode configuration, there is the ability to do iterative development and also direct benchmarking using repeatable test conditions.

8.3 Gobbies and Components

The streaming application comprises a modular set of components, each of which handles a crucial step in the fraud detection workflow. The Kafka connector used in Spark effectively transfers stream ingestion, reading information from one or more Kafka topics, and parsing the incoming JSON and Avro records (31). Explicitly defining the schema allows excellent typing and field extraction consistency. Data feature extraction is performed through a sequence of transformations applied to the streaming data frame. Metrics that these transformations can compute include the number of transactions per user within a 5-minute interval, average spending per device, or geolocation mismatches. All these real-time features get directly into the fraud scoring model.

A UDF is used to execute machine learning inference, wrapping the pre-trained model and running it on every incoming transaction. With PySpark, the model is broadcast to the cluster, so it is loaded only

once, thereby reducing overhead. This provides rapid and consistent scoring in the course of running the stream. The final step involved in the pipeline is output sink integration, which writes fraud prediction outcomes to a target destination. The predictions can be directed to Elasticsearch, where they can be indexed and viewed in real-time dashboards for use in alerting and visualization. PostgreSQL, in alternative arrangements, is considered a structured storage solution for auditing and subsequent examination.

8.4 Error Handling and Recovery

Real-time pipelines, particularly those used in the financial sector, are susceptible to fault tolerance, as data loss or temporary unavailability can have a significant impact on operations. Spark also supports checkpointing out of the box in micro-batch mode. It is a storage mechanism for the state of each batch, containing offsets from Kafka and intermediate aggregations. In the event of a failed checkpoint, work can restart at the last successful checkpoint, ensuring data consistency with minimal reprocessing.

Fault recovery in continuous processing is restricted. Spark re-executes failed tasks but does not offer as comprehensive checkpointing and state management as is provided in micro-batch mode. Additionally, support for exact-once semantics and event-time watermarks is not yet fully incorporated in the continuous mode, which is not conducive to critical applications that require deterministic processing. To reduce these constraints, continuous jobs should be closely monitored, limited to stateless transformations, and defended by external countermeasures (logging of unprocessed events or propagation of suspicious transactions to secondary auditing systems).

9. Results and Analysis

9.1 Performance Metrics Comparison

A comparison of real-time fraud detection using Micro-Batch and Continuous Processing modes in Apache Spark Structured Streaming revealed notable differences in system behavior and performance, depending on the application's specific priorities. Both modes were tested under identical system configurations and input rates to ensure a fair comparison using the same datasets and machine learning models. The choice between these modes hinges on the desired balance between throughput, latency, and processing guarantees (15). Understanding this trade-off is crucial when architecting a fraud detection pipeline for real-time responsiveness and scalability. The mode of Continuous Processing showed a considerable lead in terms of latency. Each transaction also took between 20 to 80 milliseconds to process, which allowed scoring and creation of alerts to happen nearly instantaneously. This minimal latency occurred because continuous processing was inherently efficient, with each record being processed as an independent and discrete entity, incurring minimal overhead due to scheduling or batch synchronization. By contrast, latency in Micro-Batch mode varied between 400 and 700 milliseconds. This delay is still acceptable to detect, albeit with greater latency in situations where fine-grained aggregation and multi-step fraud detection rules are necessary (13).

The other distinguishing feature was throughput. Continuous mode maintained a processing rate of about 1700 records per second, compared to 1200 records per second during the Micro-Batch mode. Continuous mode is exciting in high-frequency setups, such as real-time payments or instant e-commerce fraud checks, as the limit on the number of events communicated with every transfer is higher, and the event latency is lower. The results were well poised when evaluating the effectiveness of machine learning. The precision and recall of the Micro-Batch mode were 91.3% and 88.5%, respectively, which were slightly higher than those of the Continuous mode, at 90.8% and 87.2%, respectively. The incremental benefit noted under the Micro-Batch mode is ascribed to the enhanced context gathered using windowed features

and stateful transformations, which is the reason why the model was more capable of identifying fraudulent transactions as compared to legitimate ones. Continuous mode is not able to support more complex transformations, provided a little less, although competitive, accuracy.

Consumption of resources was as anticipated. In micro-batch mode, periodic CPU spikes were also observed, coinciding with the start and end of each mini-batch (2). These outbursts are a side effect of the organized implementation in specified periods, and uneven resource deployment over time can be a result. By contrast, Continuous Processing availed a less jittery and steadier CPU load profile. The continuous record input and the absence of batch orchestration contributed to stable performance, especially in systems with strict limitations, resulting in less variance in computations.

As shown in Table 5, while continuous mode offers lower latency and smoother resource utilization, micro-batch mode supports more complex processing, yielding slightly higher model accuracy due to the use of windowed aggregations and stateful features.

Table 5: ***Comparison of Micro-Batch and Continuous Processing Modes in Real-Time Fraud Detection***

Metric Category	Metric	Micro-Batch Mode	Continuous Mode
Latency	Average Latency	400–700 ms	20–80 ms
	Nature of Latency	Scheduled in batches; incurs synchronization overhead	Near-instant, per-record processing with minimal overhead
Throughput	Records per Second	~1200 rps	~1700 rps
ML Effectiveness	Precision	91.3%	90.8%
	Recall	88.5%	87.2%
	Notes on Accuracy	Higher accuracy due to use of windowed features	Slightly lower due to limitations in complex transformations
Resource Utilization	CPU Behavior	Periodic spikes due to batch cycles	Stable and consistent CPU usage
	System Stability	More jitter due to uneven deployment	Less jitter, better suited for systems needing consistency
Best Use Cases	Application Suitability	Good for complex rules and context-based fraud detection	Ideal for real-time payments and fast-response environments

9.2 Graphical Results

Graphical processes gave additional information on the behavior of every streaming mode. When the system latency was plotted as a time series, it was observed that, in Continuous mode, the system latency curve was always low and stable (except for occasional rises above 80 milliseconds due to short bursts of high input rate). The micro-batch mode had a stair-step appearance with evident spikes in execution every two seconds, corresponding to the set batch interval. Precision-recall analysis revealed that, although the overall performance of the models was high, the Micro-Batch mode maintained a slight

lead, particularly in offering a higher recall with slightly lower precision. This becomes particularly important in fraud detection, where failing to detect a fraudulent transaction (false negative) can be operationally costly compared to incorrectly flagging a good transaction as fraudulent (false positive). The CPU utilization over the total streaming duration, as shown in a heatmap, revealed the dynamic nature of the workload in the presence of Micro-Batch, where resource consumption spiked periodically (1). Continuous mode, on the other hand, has shown a nearly flat usage line, supporting the predictability and manageability of resource demands.

As shown in Table 6, the micro-batch mode introduces stair-step latency behavior and periodic CPU spikes, reflecting its batch-oriented design. However, it yields slightly better precision and recall, particularly in fraud detection scenarios that benefit from historical context.

Table 6: ***Graphical Analysis of Micro-Batch vs Continuous Processing***

Aspect	Micro-Batch Mode	Continuous Mode
Latency Behavior	Stair-step pattern with spikes every 2 seconds (400–700 ms latency)	Low and stable latency curve; occasional spikes above 80 ms with high input bursts
Precision-Recall	Slightly higher precision (91.3%) and recall (88.5%) due to windowed features	Slightly lower precision (90.8%) and recall (87.2%) due to limited transformation
False Negative Risk	Lower, due to better context handling	Slightly higher, may miss complex fraud patterns
CPU Utilization	Periodic spikes in resource usage due to batch cycles	Smooth, steady usage profile over time
Resource Predictability	Less predictable due to batch-based resource bursts	More predictable and manageable

9.3 Interpretations Key

The comparative analysis highlights that Continuous Processing is particularly suitable for situations where there are urgent needs for detection and rapid alerts. With its very low latency and high throughput, it is ideally suited for mission-critical applications, including transaction pre-authorization, real-time blocking of fraudulent behavior, or high-speed trading where decisions must be made in milliseconds. In contrast, Micro-Batch mode remains the architecture of choice in cases where fraud detection logic involves complex calculations, time window aggregation, or multiple streams joining. The fact that it provides support to stateful computations, recovers after failures with the help of checkpoints, and runs under well-defined scheduling makes it more dependable on systems where a high degree of accuracy and analytical depth is of more concern than immediate ones. Generally, both processing modes offer desirable functionalities, and the choice between them should reflect the specific requirements of the fraud detection strategy in terms of latency, accuracy, and transformation. A valuable and robust solution can be provided by having an architecture in which both modes can be concentrated with different components (using one architecture with Continuous mode to warn and with Micro-Batch to confirm early).

10. Practical Recommendations

10.1 When to Use Micro-Batch

The micro-batch mode features a mature, robust, and flexible stream processing model that can be developed to implement fraud detection, offering rich analytical workflows and even stateful computations. Such a mode will then become the default when the transformations involved in the use case are more sophisticated (joins between multiple data streams, multiple steps of feature engineering, or windowed aggregation, depending on the data's history). This is the basis of many fraud detection systems that rely on long-term behavioral profiling as opposed to per-event classification. Micro-batch mode provides an optimal trade-off between analytical sophistication and speed responsiveness in situations where lower latency demands are permitted, typically between several hundred milliseconds and a second. In such cases, the Micro-Batch mode offers the perfect approach. The micro-batch is utilized effectively in financial systems that aim to generate post-transactional alerts, batch investigations, or semi-real-time risk scores without compromising model accuracy or operational integrity. The support of the complete Spark SQL API in this mode, in turn, enables developers and data scientists to use the same data transformation techniques they are familiar with, resulting in a quicker and easier development process. Such elasticity is especially significant when implementing models with complex preprocessing or those that have been retrained and upgraded extensively (19).

As illustrated in Figure 8, micro-batch mode strikes a practical balance between real-time streaming and traditional batch processing, making it a valuable design choice for hybrid fraud detection systems.

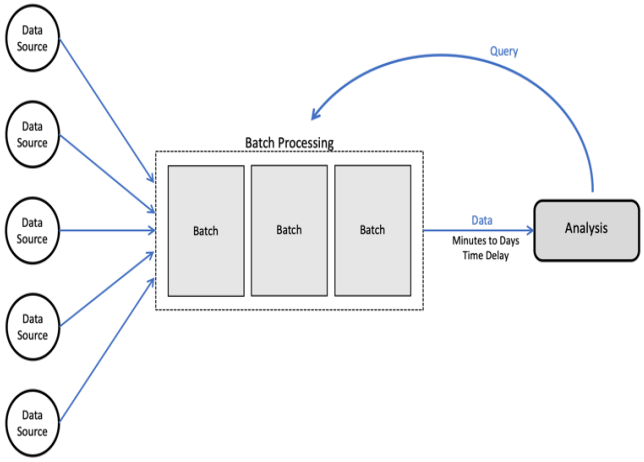

Figure 8: Batch-stream-a-cheat-sheet

10.2 The Use of the Continuous Mode

The Continuous Processing mode should be used only when mandatory ultra-low latency requirements are present. Applications that require rapid fraud detection, ideally in under 100 milliseconds, benefit from this execution model. Applications requiring fast decision-making, such as instant payment authorization, suspicious log-in detection, and automatic account lock systems, require immediate decision capabilities that are not possible on a periodic batch basis. It is best on pipelines that consist of lightweight transformations. Detection of anomalies using only simple thresholds, mismatched geolocation, or detection of blocked devices mostly aligns with the performance profile of continuous execution. Transformations are infrequently used in this mode, making it preferable to use when the logic necessary to detect fraud can be stated without the need for aggregations or joins. Continuous mode should

be used with caution in production, as it is experimental in many Spark editions (16). The trade-offs in fault tolerance, transformation coverage, and debugging tools must also be considered before it is used as the central engine of critical fraud detection systems. However, when properly configured, continuous mode offers unrivaled responsiveness and enables real-time alerting and remediation.

10.3 Best Practices

The successful operation of real-time fraud detection systems involves more than just selecting the proper processing mode. Continuous model maintenance, along with system observability, is crucial for maintaining effectiveness over time. The performance of machine learning models, especially those trained on historical data, can decline over time due to concept drift. Techniques used in fraud change fast, and the use of untrained models may render them ineffective. A schedule for retraining on time or a specific performance level of the model should be implemented to maintain high predictive performance as user behavior and attack patterns change.

The system should include monitoring tools (Prometheus and Grafana), which monitor key performance indicators in real time. System latency, system throughput, memory usage, and model inference times are meaningful metrics that can inform pipeline health and help trigger anomaly alerts. Visualization dashboards empower the operations team to react fast to any undesirable development in the streaming system. To support the retraining of the model, drift-detection modules may be placed in the streaming pipeline itself (12). The modules analyze changes in feature distributions, the prevalence of labels, and the confidence of model predictions, and warn early if current models are no longer valid. These modules help automate the lifecycle of machine learning components, enabling real-time fraud detection without requiring continual human attention. Choosing one of the Micro-Batch and Continuous Processing modes will be determined by a keen analysis of latency sets, the level of complexity of the transformation, and other factors related to the operation of the fraud detection system. A hybrid strategy that utilizes both modes to serve various layers of the detection stack, paired with robust monitoring and model governance policies, can yield a strong platform for real-time fraud detection in digital environments.

As shown in Table 7, successful implementations emphasize the importance of ongoing model maintenance, continuous monitoring and observability, and proactive detection of model drift.

Table 7: ***Best Practices for Real-Time Fraud Detection Systems***

Best Practice Area	Key Recommendations
Model Maintenance	Regularly retrain models to combat concept drift and adapt to changing fraud patterns.
Monitoring & Observability	Use tools like **Prometheus** and **Grafana** to track metrics like latency, throughput, memory, and inference times.
Drift Detection	Integrate drift-detection modules in the pipeline to monitor feature changes, label shifts, and confidence drops.
Automation & Alerts	Automate retraining triggers and system alerts using real-time anomaly detection in streaming metrics.
Processing Mode Selection	Choose **Micro-Batch** for complex processing, **Continuous** for low-latency needs, or use a **hybrid** approach.

Best Practice Area	Key Recommendations
Governance & Strategy	Combine processing strategies with strong model lifecycle management and monitoring policies.

11. Future Outlook

Although Spark Structured Streaming is a perfectly adequate framework to start with, in terms of designing effective fraud detection pipelines, several limitations continue to undermine the process of real-time ML deployment and stream processing at scale. Importantly, Continuous Processing is still an ongoing development in most Spark versions. It does not offer in-depth support for operations such as joins, aggregations, and advanced windowing, and, therefore, cannot be used in more complex fraud scenarios. Moreover, its fault recovery algorithms are not as sophisticated as those in Micro-Batch processing, which poses a concern for mission-critical applications. The other prevailing problem is that it is not easy to obtain labeled data in real time. Most fraud detection solutions are based on supervised learning and presuppose the availability of high-quality labels that must be received promptly. Fraud labels, in reality, are frequently late or noisy, making model retraining and verification challenging. Real-time systems cannot rely on existing strategies to manage this ambiguity and must instead resort to feedback information about analysts or user behavior after the event has occurred.

In the future, several directions may emerge that could lead to real-time fraud detection. The first approach combines active learning and online learning methods, enabling models to be developed in conjunction with streaming data and user input. These techniques can mitigate the need for extensive online training and allow systems to adapt to novel fraud patterns in near real-time (6). Another direction, which can also yield insights into performance trade-offs between frameworks, is to use benchmarking Spark Structured Streaming with other stream processing engines, such as Apache Flink and Kafka Streams. They could be more appropriate to the class of fraud analytics: these tools can provide various advantages in native event-time handling, state management, or operator granularity. Additionally, the application of deep learning and graph-based machine learning represents an interesting frontier. The network effect: On interconnected networks, such as those involving mule accounts, fraudulent merchant groups, or stolen credential rings, fraud is common. These complex associations are becoming more discernible by graph neural networks and deep anomaly detection models, which form a layer of intelligence on top of standard rule-based or tree-based models.

The use of hybrid systems, which dynamically alternate between processing modes according to the transaction's risk level, has the potential to increase both speed and accuracy. In continuous mode, as with low-risk transactions, scoring can be done in real-time. However, high-risk or ambiguous transactions will pass through micro-batch pipelines for more thorough analysis. These adaptive architectures have the potential to strike a balance between rapid and imposter real-time reactions and rigorous analysis, ultimately yielding better detection results and increased user confidence. To summarize, the future of real-time fraud detection lies in streaming architectures, intelligent machine learning, and orchestration. Such systems are becoming increasingly critical in the process of creating digital financial ecosystems that are resilient against constantly evolving threats, driven by the maturation of technologies and the growing volume of data.

12. Conclusion

Real-time fraud detection using machine learning in Apache Spark Structured Streaming exemplifies how a large-scale streaming processor can be leveraged with predictive capabilities to meet the demands of modern fraud systems. Driven by increases in both speed and volume, as well as the complexity of financial transactions, real-time streaming systems have surpassed traditional batch-based detection systems, as the availability of real-time streaming data necessitates immediate, data-driven responses. The proposed paper compares two main models of Spark Structured Streaming —namely, Micro-Batch and Continuous Processing —to evaluate their performance as tools for supporting fraudulent pipelines within the context of real-time constraints.

Micro-batch Processing proved to be both mature and versatile, and the paradigm was perfectly acceptable in cases where advanced analytic needs were present. It allows complex feature engineering with the support of stateful aggregations, scheduler-aware joins, and sophisticated Spark SQL functions. These are essential when detecting fraud, which relies on analyzing behavior patterns over time, i.e., deviations in spending patterns or a surge in transactions across accounts and geolocations. The trade-off is worthwhile, even with its latency, which ranges from 400 to 700 milliseconds in most experiments on the application side, where the depth of analysis trumps immediate feedback. Micro-batch also supports exceptional fault tolerance through checkpointing and recovery, which is better suited for production environments that require strength and reliability. Continuous Processing, in contrast, showed a better responsiveness. By processing the records as they arrive, the overhead of scheduling them in batches is avoided, and end-to-end latency is consistently less than 100 milliseconds. This enables it to excel in high-frequency, time-sensitive applications, such as real-time payment validation, fraud warnings during logins, or transaction denials at the point of sale. Nevertheless, its present constraints are to be considered with care. Continuous mode offers limited operations and lacks the transformation potential of Micro-Batch. Its fault recovery is still under development, and it remains experimental on many production-level Spark setups. Both modes were successful in terms of machine learning, as they enabled integration with pre-trained models for fraud scoring. Differences in precision and recall were minor, and Micro-Batch outperformed Continuous mode slightly in situations where a time-windowed feature enhanced predictive granularity. Such disparities highlight that these algorithms are just one of the factors influencing model performance, as the context and complexity of accessible features within the streaming framework can also shape and define it.

The primary lesson is that there exists no universally best processing mode. The distinct operational requirements of the intended application should guide the choice between micro-batch and continuous Processing. Where the detection logic involves complex sets of rules, aggregations, or tracking a state, Micro-Batch is more powerful and safer to use. Continuous Processing is the best response with the minimum latency when speed is of the most significant importance and the complexity of the transformation essentials is minimal. Hybrid architectures could offer the best of both worlds, as early-stage screening in Continuous mode and secondary verification or analytical enrichment in a Micro-Batch can be performed. This tiered model enables systems to optimize performance in terms of detection capabilities and technical system design based on risk management priorities. Spark Structured Streaming, in conjunction with modern machine learning (ML) techniques, offers a scalable and flexible framework for developing fraud detection systems that can adapt to emerging threats. The combination of stream processing and machine learning will be a significant pillar in the war against fraud as organizations continue to digitalize their financial operations.

References

[1] Al Jawarneh, I. M., Bellavista, P., Corradi, A., Foschini, L., & Montanari, R. (2023). SpatialSSJP: QoS-aware adaptive approximate stream-static spatial join processor. *IEEE Transactions on Parallel and Distributed Systems*, *35*(1), 73-88.

[2] Athlur, S., Saran, N., Sivathanu, M., Ramjee, R., & Kwatra, N. (2022, March). Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems* (pp. 472-487).

[3] Batani, J. (2017). An adaptive and real-time fraud detection algorithm in online transactions. *International Journal of Computer Science and Business Informatics*, *17*(2), 1-12. https://www.researchgate.net/profile/John-Batani/publication/322131441_An_Adaptive_and_Real-Time_Fraud_Detection_Algorithm_in_Online_Transactions/links/5a4678a2aca272d2945ec3dd/An-Adaptive-and-Real-Time-Fraud-Detection-Algorithm-in-Online-Transactions.pdf

[4] BATE, A. (2023). Auditable Data Provenance in Streaming Data Processing.

[5] Baud, R., Manzoori, A. R., Ijspeert, A., & Bouri, M. (2021). Review of control strategies for lower-limb exoskeletons to assist gait. *Journal of neuroengineering and rehabilitation*, *18*, 1-34. https://link.springer.com/article/10.1186/s12984-021-00906-3

[6] Bello, H. O., Ige, A. B., & Ameyaw, M. N. (2024). Adaptive machine learning models: concepts for real-time financial fraud prevention in dynamic environments. *World Journal of Advanced Engineering Technology and Sciences*, *12*(02), 021-034.

[7] Chavan, A. (2022). Importance of identifying and establishing context boundaries while migrating from monolith to microservices. Journal of Engineering and Applied Sciences Technology, 4, E168. http://doi.org/10.47363/JEAST/2022(4)E168

[8] Chavan, A. (2024). Fault-tolerant event-driven systems: Techniques and best practices. Journal of Engineering and Applied Sciences Technology, 6, E167. http://doi.org/10.47363/JEAST/2024(6)E167

[9] Dhanagari, M. R. (2024). MongoDB and data consistency: Bridging the gap between performance and reliability. *Journal of Computer Science and Technology Studies, 6*(2), 183-198. https://doi.org/10.32996/jcsts.2024.6.2.21

[10] Dhanagari, M. R. (2024). Scaling with MongoDB: Solutions for handling big data in real-time. *Journal of Computer Science and Technology Studies,* *6*(5), 246-264. https://doi.org/10.32996/jcsts.2024.6.5.20

[11] Emma, O. T., & Peace, P. (2023). Building an Automated Data Ingestion System: Leveraging Kafka Connect for Predictive Analytics.

[12] Georgiou, E. (2024). DEPLOYING ONLINE MACHINE LEARNING MODELS WITH REAL-TIME DATA PIPELINES.

[13] Hosain, M. T., Zaman, A., Abir, M. R., Akter, S., Mursalin, S., & Khan, S. S. (2024). Synchronizing object detection: applications, advancements and existing challenges. *IEEE access*. https://doi.org/10.1109/ACCESS.2024.3388889

[14] Isah, H., Abughofa, T., Mahfuz, S., Ajerla, D., Zulkernine, F., & Khan, S. (2019). A survey of distributed data stream processing frameworks. *IEEE Access*, *7*, 154300-154316.

[15] Karwa, K. (2024). The future of work for industrial and product designers: Preparing students for AI and automation trends. Identifying the skills and knowledge that will be critical for future-proofing design careers. *International Journal of Advanced Research in Engineering and Technology*, *15*(5). https://iaeme.com/MasterAdmin/Journal_uploads/IJARET/VOLUME_15_ISSUE_5/IJARET_15_05_011.pdf

[16] Khan, Z., Anjum, A., Soomro, K., & Tahir, M. A. (2015). Towards cloud based big data analytics for smart future cities. *Journal of Cloud Computing*, *4*, 1-11.

[17] Konneru, N. M. K. (2021). Integrating security into CI/CD pipelines: A DevSecOps approach with SAST, DAST, and SCA tools. *International Journal of Science and Research Archive*. Retrieved from https://ijsra.net/content/role-notification-scheduling-improving-patient

[18] Kumar, A. (2019). The convergence of predictive analytics in driving business intelligence and enhancing DevOps efficiency. International Journal of Computational Engineering and Management, 6(6), 118-142. Retrieved from https://ijcem.in/wp-content/uploads/THE-CONVERGENCE-OF-PREDICTIVE-ANALYTICS-IN-DRIVING-BUSINESS-INTELLIGENCE-AND-ENHANCING-DEVOPS-EFFICIENCY.pdf

[19] Maharana, K., Mondal, S., & Nemade, B. (2022). A review: Data pre-processing and data augmentation techniques. *Global Transitions Proceedings*, *3*(1), 91-99. https://doi.org/10.1016/j.gltp.2022.04.020

[20] Makki, S., Assaghir, Z., Taher, Y., Haque, R., Hacid, M. S., & Zeineddine, H. (2019). An experimental study with imbalanced classification approaches for credit card fraud detection. *Ieee Access*, *7*, 93010-93022. https://doi.org/10.1109/ACCESS.2019.2927266

[21] Malempati, M. (2022). Transforming Payment Ecosystems Through The Synergy Of Artificial Intelligence, Big Data Technologies, And Predictive Financial Modeling. *Big Data Technologies, And Predictive Financial Modeling (November 07, 2022)*. https://dx.doi.org/10.2139/ssrn.5246665

[22] Mehmood, E., & Anees, T. (2020). Challenges and solutions for processing real-time big data stream: a systematic literature review. *IEEE Access*, *8*, 119123-119143. https://doi.org/10.1109/ACCESS.2020.3005268

[23] Nanfack, G., Temple, P., & Frénay, B. (2022). Constraint enforcement on decision trees: A survey. *ACM Computing Surveys (CSUR)*, *54*(10s), 1-36. https://doi.org/10.1145/3506734

[24] Nyati, S. (2018). Transforming telematics in fleet management: Innovations in asset tracking, efficiency, and communication. International Journal of Science and Research (IJSR), 7(10), 1804-1810. Retrieved from https://www.ijsr.net/getabstract.php?paperid=SR24203184230

[25] Rajpurohit, A. M., Kumar, P., Kumar, R. R., & Kumar, R. (2023). A Review on Apache Spark. *Kilby*, *100*, 7th. https://dx.doi.org/10.2139/ssrn.4492445

[26] Raju, R. K. (2017). Dynamic memory inference network for natural language inference. International Journal of Science and Research (IJSR), 6(2). https://www.ijsr.net/archive/v6i2/SR24926091431.pdf

[27] Raptis, T. P., & Passarella, A. (2023). A survey on networked data streaming with apache kafka. *IEEE access*, *11*, 85333-85350. https://doi.org/10.1109/ACCESS.2023.3303810

[28] Reurink, A. (2019). Financial fraud: A literature review. *Contemporary topics in finance: A collection of literature surveys*, 79-115. https://doi.org/10.1002/9781119565178.ch4

[29] Sardana, J. (2022). Scalable systems for healthcare communication: A design perspective. *International Journal of Science and Research Archive*. https://doi.org/10.30574/ijsra.2022.7.2.0253

[30] Sardana, J. (2022). The role of notification scheduling in improving patient outcomes. *International Journal of Science and Research Archive*. Retrieved from https://ijsra.net/content/role-notification-scheduling-improving-patient

[31] Shetty, S. (2019). *Improving processing of real-time Big Data in Smart Grids using Apache Flink and Kafka* (Doctoral dissertation, Dublin, National College of Ireland).

[32] Singh, V. (2021). Generative AI in medical diagnostics: Utilizing generative models to create synthetic medical data for training diagnostic algorithms. International Journal of Computer Engineering and Medical Technologies. https://ijcem.in/wp-content/uploads/GENERATIVE-AI-IN-MEDICAL-DIAGNOSTICS-UTILIZING-GENERATIVE-MODELS-TO-CREATE-SYNTHETIC-MEDICAL-DATA-FOR-TRAINING-DIAGNOSTIC-ALGORITHMS.pdf

[33] Singh, V. (2022). Visual question answering using transformer architectures: Applying transformer models to improve performance in VQA tasks. Journal of Artificial Intelligence and Cognitive Computing, 1(E228). https://doi.org/10.47363/JAICC/2022(1)E228

[34] Sukhadiya, J., Pandya, H., & Singh, V. (2018). Comparison of Image Captioning Methods. *INTERNATIONAL JOURNAL OF ENGINEERING DEVELOPMENT AND RESEARCH*, *6*(4), 43-48. https://rjwave.org/ijedr/papers/IJEDR1804011.pdf

[35] Zahra, F. T., Bostanci, Y. S., Tokgozlu, O., Turkoglu, M., & Soyturk, M. (2024). Big Data Streaming and Data Analytics Infrastructure for Efficient AI-Based Processing. In *Recent Advances in Microelectronics Reliability: Contributions from the European ECSEL JU project iRel40* (pp. 213-249). Cham: Springer International Publishing. https://link.springer.com/chapter/10.1007/978-3-031-59361-1_9