INTERNATIONAL JOURNAL OF
INTELLIGENT OPTIMIZATION
AND LOGISTICS

Available online at www.ijiol.com

RESEARCH ARTICLE

Streaming Fraud Detection: A Kafka-Spark Machine Learning Approach for Banks

Redouane Hakimi[a,*], **Youssef Tajani**[a], **Mohamed Ouzineb**[a], **Karim El Moutaouakil**[b]

[a] Institut National de Statistique et d'Économie Appliquée, Rabat, Morocco

[b] Université Sidi Mohamed Ben Abdellah, Fes, Morocco

ABSTRACT

The growth of digital payments requires fraud detection systems that can process high-velocity transaction streams while keeping the number of false alarms manageable. This research proposes a real-time solution that uses Apache Kafka for data streaming, Apache Spark for distributed processing and Spark MLlib for machine learning (ML). Unlike traditional batch methods, the proposed system combines batch and real-time processing layers in a Lambda architecture, deployed as Docker micro-services. The system is evaluated on confidential card-transaction data from a Moroccan bank with an imbalance ratio of 675:1. Decision Trees (DT), Random Forests (RF) and Gradient-Boosted Trees (GBT) were trained on data rebalanced with a hybrid resampling strategy (random under-sampling and SMOTE) and tuned with the Tree-structured Parzen Estimator (TPE) using G-means as the objective. TPE reduced tuning time by 80% relative to exhaustive grid search for DT, and made the tuning of RF and GBT feasible where grid search could not be completed. The TPE-tuned Random Forest achieved the best trade-off (precision 0.343, recall 0.781, PR AUC 0.612, MCC 0.516) and about 91% fewer false alarms than its default configuration. End-to-end latency ranged from 3.5 to 5 seconds. The operational cost of the remaining false positives and the limitations of the study are discussed.

KEYWORDS: Apache Kafka · Apache Spark · Fraud Detection · Machine Learning · Banking Security · Real-Time Processing

1. Introduction

The exponential growth of digital financial transactions demands robust, real-time fraud detection systems that combine high accuracy with minimal latency. Traditional batch-processing methods have become inadequate against increasingly sophisticated cyber threats, creating an urgent need for scalable streaming solutions. This paper presents an advanced fraud detection system using a streaming architecture with Apache Kafka for data ingestion, Apache Spark for distributed processing and Spark MLlib for ML modeling.

* Corresponding author: r.hakimi@insea.ac.ma.

Article history: Received 21 September 2026; Revised 18 October 2026; Accepted 6 November 2026; Published 13 November 2026.
Publication details: Volume 1, Issue 1, Article 0002.

© 2026 The Authors. This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0).

A critical challenge in financial fraud detection involves handling severely imbalanced datasets [1, 2], where fraudulent transactions may represent less than 0.1% of all transactions [3]. This imbalance skews model performance, as demonstrated by [4] in their comparison of Poisson processes with traditional ML methods. Recent approaches like those in [5] have shown that combining ensemble methods with smart sampling techniques can significantly improve detection rates for minority fraud classes.

The class imbalance problem becomes particularly acute in real-time systems, where traditional oversampling methods may introduce unacceptable latency [6]. As highlighted in [7], feature engineering and graph-based representations offer promising alternatives to address imbalance without compromising processing speed. Our work builds upon these insights while addressing the unique constraints of streaming architectures.

Current fraud detection systems face three fundamental challenges: unacceptable detection delays, inflexibility against evolving fraud patterns and limited scalability with growing transaction volumes. While streaming technologies offer real-time processing capabilities and ML provides pattern recognition, their effective integration presents technical hurdles including computational efficiency, class imbalance and latency constraints in production environments.

Our research focuses on optimizing ML models under these constraints, using G-means, a metric suited to the minority class, as the optimization objective. The best configurations reach a G-means of about 0.89 and a balanced accuracy of up to 0.90. A key finding is the trade-off between detection performance and computational efficiency: exhaustive hyperparameter search becomes infeasible for tree ensembles on limited resources, which highlights the need for a balanced choice of algorithms and tuning strategies in operational systems.

Most published fraud detection models are developed and evaluated offline, in batch mode, often on the same public benchmark [8–10], and these studies rarely address deployment on a live transaction stream, retraining or end-to-end latency. Conversely, streaming platforms such as Kafka-ML [11, 12] and cloud Spark architectures [13, 14] provide generic infrastructure but do not address the specificities of fraud (extreme imbalance, evolving patterns, cost of false alarms). The few studies that combine both aspects, such as SCARFF [15] or recent Kafka-based comparisons [16], do not report how model hyperparameters are tuned or what tuning costs in a distributed setting.

This paper therefore addresses the following question: *can an open-source Kafka–Spark pipeline, combined with imbalance-aware resampling and Bayesian hyperparameter optimization, deliver interpretable fraud detection models with acceptable detection quality, tuning cost and latency on real banking data?*

The main contributions of this paper are:

(i) an end-to-end, containerised Lambda architecture (Kafka, Spark Streaming, Spark MLlib, HDFS, HBase, Streamlit) that decouples model training on historical data from real-time scoring (Section 3),

(ii) the integration of TPE-based hyperparameter optimization with Spark MLlib using an imbalance-aware objective (G-means), and a systematic comparison with default settings and exhaustive grid search for three interpretable tree-based models, including the tuning cost (Sections 3.8 and 5),

(iii) an empirical evaluation on confidential data from a Moroccan bank with a 675:1 imbalance ratio, a setting rarely reported in the literature,

(iv) an operational analysis of false alarms, end-to-end latency and resource usage, with an explicit statement of the limitations of the prototype (Sections 5.4.1 and 5.5).

The paper is structured as follows: Section 2 introduces banking fraud and reviews the relevant literature and existing methodologies. Section 3 details the methodology for Real-Time Data Processing. Section 4 presents the experimental design, including datasets, preprocessing and evaluation metrics. Section 5 presents results comparing our model against baselines. Section 6 concludes the paper.

2. Background and literature review

Recent advances in fraud detection have leveraged diverse methodologies across multiple domains, reflecting the growing complexity of fraudulent activities in digital ecosystems. In the domain of click fraud detection, [6]

introduced CFXGB, an innovative hybrid model that integrates Cascaded Forests with XGBoost to enhance feature transformation and classification performance. Their work demonstrated superior results on benchmark datasets while investigating the impact of parent node values in DT's, providing valuable insights for feature engineering in fraud detection systems.

For network security applications, significant progress has been made in both intrusion detection and bot identification. [17] developed a novel hybrid deep learning approach for Software-Defined Networks, combining CNNs with their SD-Reg regularization technique to mitigate overfitting while maintaining detection accuracy. Similarly, [18] proposed NNSEQ, a two-stage classifier combining neural networks with sequential probability ratio testing for real-time web bot detection, demonstrating exceptional performance in early identification of malicious sessions.

Financial fraud detection has seen particularly innovative approaches across multiple subdomains. [3] conducted a comprehensive comparison of biologically inspired HTM-CLA against traditional Artificial Neural Network architectures for credit card fraud, revealing the potential of continual learning systems. Complementing this work, [4] explored Poisson Process Modeling (PPM) as an alternative to ML methods, suggesting potential synergies between statistical and algorithmic approaches. In banking transaction analysis, [5] demonstrated how ensemble methods combined with autoencoder-based dimensionality reduction in SAS Viya environments can significantly improve detection accuracy, particularly when incorporating geographical transaction characteristics.

Emerging decentralized systems have introduced new fraud detection challenges that require innovative solutions. [19] addressed this through ConTrib, a tamper-evident ledger system for decentralized applications that successfully scaled to 10,000 peers while maintaining fraud detection within seconds. Their two-year deployment in Tribler provided compelling evidence for the system's effectiveness in real-world conditions.

Text-based financial fraud detection has also seen notable advancements. [20] shifted focus from traditional fraud detection to negative event extraction in Chinese financial news, developing a Q&A-based method that achieved high precision with reduced annotation requirements. For financial statement analysis, [7] introduced a knowledge graph approach that models feature correlations, demonstrating improved classification performance through enriched document representations.

Graph-based techniques have proven particularly valuable across multiple fraud detection contexts. [21] enhanced community detection in attributed networks through their modified label propagation algorithm, which incorporated link strength and node attributes while maintaining near linear time complexity, an important consideration for large-scale fraud detection systems.

Several advances in distributed ML systems have focused on optimizing three critical aspects: computational efficiency across heterogeneous environments, infrastructure flexibility and semantic-aware data processing. These developments provide valuable foundations for building robust fraud detection systems capable of handling real-time financial data streams. A key breakthrough in computational efficiency was achieved by [12] through their Kafka-ML extension for Distributed Deep Neural Networks. By combining BranchyNet architectures with containerized deployment (Docker/Kubernetes) and Apache Kafka for stream processing, they demonstrated how early-exit inference strategies could dramatically reduce latency in edge-cloud environments. This work is particularly relevant for fraud detection systems that require low-latency processing of transaction streams while maintaining high accuracy.

Building on these distributed computing principles, [13] addressed the infrastructure flexibility challenge through their cloud-agnostic reference architectures. Their integration of Apache Spark with the Occopus orchestrator enables automated provisioning of complete ML environments across multiple cloud providers. This approach solves the critical portability issues faced by financial institutions operating in multi-cloud environments.

Complementing these infrastructure-focused solutions, [14] made significant contributions at the data processing level through their TITAN platform. Their semantic-enhanced workflow management system, powered by the BIGOWL ontology and integrating Kafka/Avro/Spark, ensures data quality and domain knowledge preservation throughout the analytics life-cycle. For fraud detection applications, such semantic frameworks could enable more sophisticated pattern recognition by incorporating financial domain knowledge directly into the processing pipeline.

2.1. Banking fraud: definitions, types and challenges

Fraud can be defined as the use of deception to obtain an unjust financial advantage or to cause a loss to another party [22]. In retail banking, the main types are [8, 22]:

- **Card-present fraud**: transactions made with a lost, stolen or counterfeit (skimmed) card at a terminal or ATM,
- **Card-not-present (CNP) fraud**: use of stolen card details for remote payments (e-commerce, mail or telephone orders),
- **Account takeover**: a fraudster gains control of a customer's account or credentials (e.g. through phishing) and initiates payments,
- **Application (identity) fraud**: opening accounts or obtaining cards and loans with false or stolen identities,
- **Internal fraud and money laundering**: abuse by insiders and concealment of illicit funds through chains of transactions.

The system proposed in this paper addresses *transaction-level payment-card fraud*: each incoming card transaction is classified as legitimate or fraudulent from its attributes and from behavioural aggregates. This covers the first three types insofar as they materialise as unauthorised card transactions, whereas application fraud, internal fraud and money laundering rely on other data and are outside the scope of this work.

Fraud detection differs from a standard classification problem because of several challenges [8, 22, 23]: (a) *class imbalance*, which makes accuracy misleading [24, 25], (b) *concept drift*, since customer behaviour and fraud strategies evolve [26], (c) *verification latency*, since true labels are only known after investigation [23], (d) *real-time constraints*, requiring low-latency scalable infrastructure [15], (e) the *cost of errors and the alert budget*, since false negatives cause direct losses while false positives generate analyst workload and customer friction [1, 23], and (f) *explainability and regulation*, which favour interpretable models [27]. Our design addresses (a) through hybrid resampling and imbalance-aware metrics, (d) through the Kafka–Spark architecture, (e) through an explicit analysis of false positives and (f) through tree-based models, while (b) and (c) are only partially addressed through the decoupling of training and serving, and are discussed as limitations.

2.2. Machine learning models for fraud detection

Reviews of the field [8, 22] show that supervised learning dominates, with logistic regression, decision trees, random forests, gradient boosting, support vector machines (SVM) and neural networks as the most frequently used models. Random Forests and gradient boosting are consistently among the strongest models on tabular credit data [28], and Random Forests are at the core of realistic FDS operating on tens of millions of real transactions [23]. LightGBM combined with Bayesian optimization has been proposed by Taha and Malebary [10]. SVMs are competitive on small or balanced samples but their training cost grows quickly with data size [9]. Neural approaches range from multilayer perceptrons and convolutional networks [9] to recurrent models exploiting transaction sequences, and Mienye and Jere [29] review these deep learning methods and point out their data requirements, computational cost and limited interpretability. Behavioural aggregates computed over time windows are also a key ingredient of effective FDS [23].

2.3. Handling class imbalance

Imbalance can be addressed at the data level (resampling), at the algorithm level (cost-sensitive learning) or with ensembles [1, 2, 24]. Random under-sampling reduces training cost but discards information and distorts posterior probabilities [30]. SMOTE [31] generates synthetic minority examples by interpolation between neighbouring minority instances, and many variants exist [32]. Combining over- and under-sampling is a common hybrid strategy [2, 32]. A known side effect of over-sampling is a shift of the decision boundary towards the minority class, which increases recall at the expense of precision [30, 32]. Imbalance-aware measures such as PR AUC, G-means and MCC are recommended over accuracy and ROC AUC [33, 34].

2.4. Hyperparameter optimization methods

Hyperparameter optimization (HPO) methods include: *grid search*, exhaustive but exponential in the number of hyperparameters, *random search*, often more efficient than grid search [35], *Bayesian optimization* with Gaussian processes or with the Tree-structured Parzen Estimator (TPE) [36], implemented in libraries such as Hyperopt and Optuna, and *metaheuristics* such as genetic algorithms or simulated annealing [3]. We use TPE because (i) it natively handles mixed and conditional search spaces (integer depths, categorical impurity, continuous learning rates), (ii) its overhead is negligible compared with training a Spark model, (iii) unlike random search, it concentrates the budget on promising regions, and (iv) Hyperopt integrates directly with PySpark. Grid search is kept as a reference wherever it is computationally feasible.

2.5. Streaming architectures for fraud detection

Apache Kafka and Apache Spark [37], with its MLlib library and Structured Streaming engine, are widely used for distributed stream processing [11, 13], and the Lambda architecture combines a batch layer for model computation with a low-latency speed layer. Among fraud-specific systems, SCARFF [15] integrates Kafka, Spark and Cassandra with a learning strategy that handles imbalance, non-stationarity and feedback latency. Daksa and Kemala [16] compare Spark and Flink in a Kafka pipeline with a pretrained Random Forest and report average latencies of 0.8–0.9 s for Spark at 10–40 transactions per second.

2.6. Positioning of the present work

Table 1 compares the closest studies with ours. Three gaps emerge: (i) studies that tune models carefully [9, 10] work offline on public benchmarks, whereas streaming systems [15, 16] do not report how hyperparameters are chosen or what tuning costs, (ii) few studies use real data from banks outside Europe, and to our knowledge none uses data from Moroccan banks, (iii) operational indicators (false alarms per detected fraud, latency, resources) are rarely reported together with predictive metrics. Our work addresses these gaps within a single open-source stack. We do not claim a new learning algorithm, as the contribution lies in the integration, the systematic comparison of tuning strategies under real resource constraints, and the evaluation on real banking data.

Table 1. Positioning with respect to closely related work (✓: addressed, –: not a focus of the study).

Study	Data	Stream.	Distrib.	Imbalance	HPO	Latency
Dal Pozzolo et al. [23]	Real, European cards	✓	–	Under-sampling, ensembles	–	–
SCARFF [15]	Real, European cards	✓	✓	Under-sampling (balanced RF)	–	–
Taha & Malebary [10]	Public benchmarks	–	–	–	Bayesian	–
Alarfaj et al. [9]	Public (European)	–	–	Discussed	–	–
Kafka-ML [11]	Generic (not fraud)	✓	✓	–	–	–
Daksa & Kemala [16]	Fraud (offline RF)	✓	✓	–	–	✓
This work	**Real, Moroccan bank**	✓	✓	**SMOTE + under**	**TPE vs. grid**	✓

3. Methodology for Real-Time Data Processing

The proposed fraud detection system is designed as a hybrid batch and real-time data processing architecture that integrates Apache Kafka, Apache Spark, Spark MLlib, Hadoop Distributed File System (HDFS), HBase, and Streamlit. As illustrated in Figure 1, the architecture combines historical data processing for machine learning model development with continuous stream processing for real-time fraud detection. The overall methodology consists of five main stages: data acquisition, data storage and streaming, real-time processing, model training and deployment, and prediction storage and visualization.

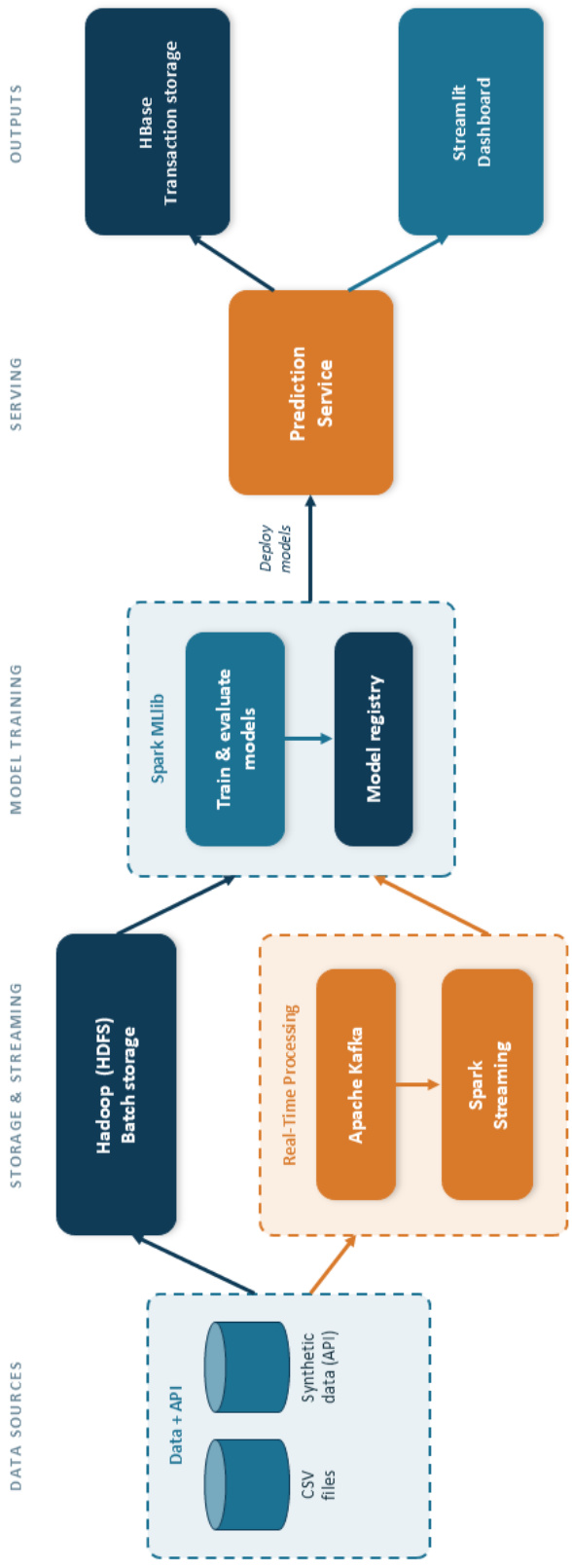


Figure 1. Architecture of the proposed fraud detection system. Dark blue: batch path (historical data, model training and registry), orange: real-time path (ingestion, stream processing and scoring).

3.1. Data Acquisition and Storage

The proposed architecture receives data from two complementary sources: historical CSV files and synthetically generated data provided through an Application Programming Interface (API). Historical data are used for batch processing, model development, and evaluation, whereas synthetically generated data are used to simulate continuously arriving financial transactions.

The two data paths are integrated within the same architecture. Historical data are stored in the Hadoop Distributed File System (HDFS), which provides distributed storage for the datasets used during model development. In contrast, newly generated transactions are transmitted to Apache Kafka for real-time processing. This separation enables the system to simultaneously exploit historical information for model training and process newly arriving transactions for real-time prediction.

3.2. Event-Streaming Architecture Using Kafka

In the proposed fraud detection pipeline, Apache Kafka provides the real-time data streaming backbone between the data sources and the stream processing layer. Kafka enables the continuous ingestion, buffering, distribution, and consumption of financial transaction events.

As detailed in Table 2, Kafka Topics are used as dedicated communication channels for transaction data and fraud alerts. Producers represent the upstream systems responsible for publishing transaction events, while Consumers retrieve these events for downstream analysis. In the proposed architecture, Apache Spark Streaming acts as the primary consumer of incoming transaction data. Kafka Brokers manage message storage, partitioning, and distribution, while Consumer Groups enable multiple consumer instances to process data in parallel.

This organization allows the streaming infrastructure to distribute the processing workload across multiple instances and provides scalability when the transaction volume increases.

Table 2. Role of Kafka components in the proposed fraud detection architecture.

Component	Role
Topics	• Provide communication channels for transaction events. • Separate transaction and alert streams when required.
Producer	• Publishes transaction events to Kafka topics. • Represents the upstream transactional or data-generating systems.
Consumer	• Retrieves messages from Kafka topics. • Provides transaction streams to Spark Streaming for analysis.
Broker	• Manages message storage, partitioning, and distribution. • Supports scalable and fault-tolerant message delivery.
Consumer Group	• Distributes the processing workload among consumers. • Enables parallel processing and improves fault tolerance.

3.3. Real-Time Data Processing Using Spark Streaming

Once financial transactions are published to Kafka, Apache Spark Streaming consumes the incoming events and performs the required processing operations. The streaming layer is responsible for transforming raw transaction messages into structured feature vectors that can be processed by the deployed machine learning model.

The real-time processing pipeline includes data parsing, schema validation, data cleaning, missing-value handling, feature transformation, and feature extraction. The resulting representation is then passed to the prediction service for fraud classification.

The real-time processing workflow can be summarized as follows:

$$D_t \rightarrow \text{Kafka} \rightarrow \text{Spark Streaming} \rightarrow X_t \rightarrow \text{Prediction Service}, \tag{1}$$

where:

- D_t denotes an incoming transaction at time t.
- X_t represents the processed feature vector.

This streaming-based approach allows transactions to be processed continuously as they arrive, rather than waiting for the complete data set to become available.

3.4. Machine Learning Model Training and Evaluation

In parallel with real-time processing, historical transaction data stored in HDFS are used to develop the fraud detection model. Apache Spark MLlib provides the distributed machine learning framework for training and evaluating the predictive models.

The model development pipeline consists of data preprocessing, feature engineering, model training, and model evaluation. The historical dataset is first loaded from HDFS and transformed into a suitable representation for machine learning. Relevant transaction attributes are selected and transformed into numerical features, after which the dataset is divided into training and evaluation subsets.

The training process can be represented as:

$$M = \mathcal{T}(D_{\text{train}}), \tag{2}$$

where D_{train} denotes the training dataset, $\mathcal{T}$ represents the learning algorithm implemented using Spark MLlib, and M denotes the resulting trained model.

The trained model is evaluated using the selected performance metrics. Following evaluation, the model is stored in the model registry and subsequently made available to the prediction service.

3.5. Model Deployment and Real-Time Fraud Detection

The selected machine learning model is deployed through the prediction service. This component receives the feature vectors generated by the Spark Streaming layer and applies the trained model to each incoming transaction.

For an incoming transaction represented by the feature vector X_t, the prediction is defined as:

$$\hat{y}_t = M(X_t), \tag{3}$$

where, M is the deployed fraud detection model and $\hat{y}_t$, represents the predicted class for transaction t.

The separation between model training and real-time inference allows the prediction service to operate independently of the model training process. Consequently, new model versions can be trained and evaluated using historical data and subsequently deployed without changing the Kafka-based data ingestion mechanism.

3.6. Prediction Storage and Visualization

The output of the prediction service is directed to two complementary components: HBase and the Streamlit dashboard.

HBase provides persistent storage for transaction information and generated predictions. The stored results can subsequently be retrieved for analysis, auditing, or historical monitoring.

In parallel, the prediction results are presented through a Streamlit dashboard. The dashboard provides an interactive interface for monitoring the real-time operation of the fraud detection system and visualizing relevant transaction and prediction information.

The complete real-time inference workflow can therefore be expressed as:

$$\text{Transaction} \rightarrow \text{Kafka} \rightarrow \text{Spark Streaming} \rightarrow \text{Prediction} \rightarrow \begin{cases} \text{HBase}, \\ \text{Streamlit}. \end{cases} \tag{4}$$

3.7. Integrated System Architecture

The proposed architecture implements a modular separation of concerns between data acquisition, distributed processing, machine learning, and output management. The overall workflow integrates the batch and real-time paths as follows:

$$\begin{aligned} D_{\text{historical}} &\rightarrow \text{HDFS} \rightarrow \text{Spark MLlib} \rightarrow \text{Model Registry}, \\ D_{\text{stream}} &\rightarrow \text{Kafka} \rightarrow \text{Spark Streaming} \rightarrow \text{Prediction Service} \\ &\rightarrow \{\text{HBase}, \text{Streamlit}\}. \end{aligned} \tag{5}$$

The batch path is therefore responsible for historical data storage, model training, and model evaluation, whereas the real-time path is responsible for continuous transaction ingestion, stream processing, and fraud prediction.

This modular design provides several advantages. First, distributed storage and processing facilitate the management of increasing data volumes. Second, the decoupling of data ingestion, processing, and prediction allows individual components to be developed and maintained independently. Third, Kafka's partitioning and consumer-group mechanisms support parallel processing of incoming transaction streams. Finally, the combination of persistent storage and interactive visualization provides both historical traceability and real-time monitoring of prediction results.

Overall, the proposed methodology establishes an end-to-end pipeline that transforms raw transaction data into real-time fraud predictions, while maintaining a separate batch-processing path for continuous machine learning model development and deployment.

3.8. Hyperparameter optimization using tree-structured parzen estimator

Hyperparameters are settings that are not learned from the data, and the search space contains all candidate combinations, and its size governs the cost of the search. The alternatives to TPE and the reasons for its choice are discussed in Section 2.4.

Let $\mathcal{X}$ be the search space and $f(x)$ the loss of configuration x. We minimise $f(x) = 1 - \text{G-means}(x)$, computed on held-out data. TPE [36] models the density of configurations rather than the loss itself: past observations are split at a quantile f^* of the observed losses into "good" and "bad" groups, with densities $l(x) = p(x \mid f(x) < f^*)$ and $g(x) = p(x \mid f(x) \geq f^*)$, and the next configuration maximises $l(x)/g(x)$, which is proportional to the expected improvement. Algorithm 1 summarises the procedure as applied to Spark MLlib. The overhead of TPE is negligible compared with the cost C_f of training and evaluating one Spark model, so the total cost is approximately $N \cdot C_f$ for N evaluations, versus $|\mathcal{X}| \cdot C_f$ for exhaustive grid search over $|\mathcal{X}|$ combinations.

Algorithm 1 TPE-based hyperparameter optimization of a Spark MLlib model

Require: search space $\mathcal{X}$, budget N, start-up trials n_0, quantile γ

$H \leftarrow \emptyset$
for $i = 1$ to N **do**
  if $i \leq n_0$ **then**
    sample x_i at random from the priors on $\mathcal{X}$
  else
    $f^* \leftarrow \gamma$-quantile of the losses in H
    fit $l(x)$ on $\{x : (x, f) \in H, f < f^*\}$ and $g(x)$ on the others
    $x_i \leftarrow \arg\max_x l(x)/g(x)$ among candidates drawn from l
  end if
  train the Spark MLlib model with x_i on the rebalanced training set
  $f_i \leftarrow 1 -$ G-means on held-out data, $\quad H \leftarrow H \cup \{(x_i, f_i)\}$
end for
return $x^* = \arg\min_{(x,f)\in H} f$

4. Experimental design

4.1. Datasets

The experimental dataset consists of confidential financial transaction records provided by a Moroccan banking institution for developing real-time fraud detection systems. Due to strict non-disclosure agreements and data protection regulations, specific details about the dataset cannot be disclosed. The raw data, the exact feature definitions and the collection period therefore cannot be published, and Table 3 reports the quantitative information that can be disclosed. The records contain three groups of attributes: (i) transaction attributes (amount, date and time), (ii) card and customer attributes (card brand, card type, customer age and gender), and (iii) engineered behavioural indicators computed as time-windowed aggregates of the recent activity of the card. The anonymized transaction records include multiple engineered features that serve as behavioral indicators for fraudulent activities, while preserving customer privacy and complying with financial data security requirements.

A notable characteristic of this dataset is its severe class imbalance, with non-fraudulent transactions outnumbering fraudulent ones by a ratio of 675:1, as shown in Table 4. Exploratory Data Analysis (EDA) visualizations reveal distributions across various dimensions including age, transaction amounts stratified by age, card brand, card type and gender.

Table 3. Quantitative description of the data set.

Characteristic	Value
Source	Card transactions, Moroccan bank (confidential, anonymised)
Data volume	1.52 GiB
Initial imbalance ratio (legitimate : fraudulent)	675.13 : 1 (fraud rate $\approx 0.15\%$)
Test set	297 fraudulent transactions, original class distribution kept ($\approx 2.0 \times 10^5$ legitimate transactions)
Training set after resampling	68,790 legitimate + 13,758 fraudulent = 82,548
Attribute groups	Transaction, card/customer, behavioural aggregates

Table 4. Summary of Class Imbalance Treatment

Metric	Value
Initial Imbalance Ratio	675.13:1
Maximum Allowed Threshold	5:1
Applied Approach	Hybrid (Under-sampling + Over-sampling)
Majority Class Size (Target)	68,790
Minority Class Size (Target)	13,758
Final Approximate Imbalance Ratio	5.00:1

To mitigate the class imbalance, we applied a hybrid resampling technique combining under-sampling of the majority class with over-sampling of the minority class (SMOTE [31]), thereby enhancing model performance in fraud detection. The processed dataset is subsequently integrated into an Apache Kafka pipeline to facilitate real-time fraud prevention within a robust and reliable system architecture.

4.2. Experimental protocol

The system trains multiple model variants using Hyperopt for automated hyperparameter tuning, searching through configurations of tree depth, feature subsets and learning rates. To address the inherent class imbalance in fraud datasets, we implement a hybrid sampling approach combining SMOTE oversampling of the minority class with strategic under-sampling of the majority class. This preprocessing occurs in the batch layer before model training while maintaining temporal splits to prevent data leakage. Concretely, the data are split chronologically: models are trained on earlier transactions and evaluated on later ones, which mimics deployment [23]. Resampling is applied *only to the training set*: the majority class is randomly under-sampled to 68,790 transactions and the minority class is over-sampled with SMOTE to 13,758 transactions (ratio 5:1, Table 4). The test set keeps the original class distribution, so that the reported precision and false-positive rates reflect operating conditions. A moderate 5:1 target was preferred to full 1:1 balance in order to limit the number of synthetic examples and the resulting shift of the decision boundary [32].

The Lambda architecture separates processing into distinct batch and speed layers to balance accuracy with latency requirements. The batch layer handles feature engineering and model retraining on historical data using 20% of the total compute resources, while the speed layer uses 80% of resources for real-time inference. Feature selection occurs through iterative analysis of Gini importance scores and out-of-bag error reduction across multiple training runs. Transaction attributes are transformed into time-windowed aggregates and behavioral patterns that serve as model inputs.

Containerization through Docker packages all pipeline components including Kafka producers/consumers, Spark jobs and model servers into portable units. We orchestrate these microservices through Git-managed deployment scripts that enable rolling updates without pipeline interruption. The Streamlit monitoring dashboard connects directly to Spark's structured streaming output, providing real-time visibility into system throughput and prediction distributions.

The study employed three tree-based machine learning approaches for fraud detection: DT as the baseline model, RF as a robust ensemble method and GBT as an advanced sequential learning technique (see Table 5 for model specifications). The full search spaces contain 768 (DT), 1,152 (RF) and 6,912 (GBT) combinations (Table 6), which were explored exhaustively by grid search where feasible, and with TPE using 25 evaluations for DT and RF and 20 for GBT. The tree-based architecture was specifically chosen for its interpretability in financial fraud scenarios, with ensemble methods (RF and GBT) providing enhanced accuracy through different aggregation strategies - parallel tree construction in RF versus sequential error correction in GBT.

Table 5. Machine learning models.

Model	Description
DT	A tree-based model that makes decisions by recursively splitting the input space into regions based on feature values. Hyperparameters were optimized over 768 combinations to balance model complexity and performance.
RF	An ensemble model that constructs multiple DT's and aggregates their predictions to improve accuracy and robustness. Hyperparameter optimization explored 1,152 combinations.
GBT	An ensemble method that builds DT's sequentially, with each tree correcting errors of the previous ones. Hyperparameters were tuned across 6,912 combinations to optimize performance.

Each model is trained under three tuning strategies, which are compared in Section 5: **S-** (standard), with the default Spark MLlib hyperparameters, **C-** (complete grid), with exhaustive grid search over the search space of Table 6, and **H-** (Hyperopt), with TPE over the same space (Section 3.8), minimising $1 - \text{G-means}$ on held-out data. All models use the default decision threshold of 0.5 on the predicted probability. The exhaustive searches for RF and GBT did not complete (execution error after more than 400 min for C-RF, more than 550 min for C-GBT).

Table 6. Hyperparameters tuned for each model (search space shared by grid search and TPE, the selected values are given in Table 8).

Hyperparameter (Spark MLlib)	DT	RF	GBT
`maxDepth`	✓	✓	✓
`maxBins`	✓	✓	✓
`impurity` (gini / entropy)	✓	✓	–
`minInfoGain`	✓	✓	✓
`minInstancesPerNode`	✓	✓	✓
`numTrees` / `maxIter`	–	✓	✓
`stepSize` (learning rate)	–	–	✓
`subsamplingRate`	–	✓	✓
Grid size (C-)	768	1,152	6,912
TPE evaluations (H-)	25	25	20

4.3. Evaluation metrics

Let TP, FP, TN and FN denote the entries of the confusion matrix, with fraud as the positive class. Accuracy is not used as a criterion because it is misleading under imbalance [25, 33]: a classifier that predicts "legitimate" for every transaction would reach 99.85% on our test set. Table 7 lists the metrics used, G-means and PR AUC are the primary criteria, and MCC is reported as a balanced single-number summary [34]. All models are evaluated on the same test set, which contains 297 actual fraudulent transactions and keeps the original class distribution.

Table 7. Evaluation metrics.

Metric	Definition
Precision	$\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$
Recall (sensitivity)	$\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$
Specificity	$\mathrm{TN}/(\mathrm{TN}+\mathrm{FP})$, false-positive rate = $1-\mathrm{Specificity}$
F1	$2\,\mathrm{Precision}\cdot\mathrm{Recall}/(\mathrm{Precision}+\mathrm{Recall})$
Balanced accuracy	$(\mathrm{Recall}+\mathrm{Specificity})/2$
G-means	$\sqrt{\mathrm{Recall}\cdot\mathrm{Specificity}}$
MCC	$\dfrac{\mathrm{TP}\cdot\mathrm{TN}-\mathrm{FP}\cdot\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$
ROC AUC / PR AUC	Area under the ROC / precision–recall curve

5. Results and discussion

5.1. Hyperparameter search space and optimal hyperparameters

The total number of combinations tested for DT, RF and GBT was 768, 1,152 and 6,912 respectively, ensuring a comprehensive exploration of the parameter space. The hyperparameter optimization results reveal significant differences between the exhaustive search and Hyperopt-tuned models. The optimal hyperparameters identified for each model after exploration are summarized in Table 8.

Table 8. Summary of optimal hyperparameters for all tested models

	Decision Tree (768 comb.)	Decision Tree (Hyperopt)	Random Forest (Hyperopt)	GBT (Hyperopt)
Maximum Depth	20	10	30	10
Maximum Bins	256	256	128	256
Impurity	Entropy	gini	Entropy	–
Min Info Gain	0	0.0000	0.0000	0.0729
Min Instances per Node	1	2	1	5
Number of Trees/Iterations	–	–	20	50
Learning Rate	–	–	–	0.0209
Sampling Rate	–	–	1.00	0.8200

"–": the parameter is not applicable to the respective model.

The Hyperopt-optimized DT is shallower than the tree obtained by exhaustive search (maximum depth 10 instead of 20), with the same number of bins (256), no minimum information gain and a slightly larger minimum node size (2 instead of 1). These constraints yield a tree that is easier to inspect and less prone to over-fitting.

For ensemble methods, Hyperopt revealed significantly different optimal configurations. The Random Forest model achieves a balance between complexity and predictive power using 20 trees with a maximum depth of 30, entropy-based impurity and a full sampling rate of 1.00. In contrast, the GBT approach leverages 50 iterations of shallower trees with a depth of 10, a finely tuned learning rate of 0.0209, a minimum information gain of 0.0729 and a subsampling rate of 0.8200. These settings illustrate the ability of Hyperopt to adapt the hyperparameters to the characteristics of each algorithm.

5.2. Optimization benefits

The Hyperopt approach yields models that are:

- More interpretable (simpler structures).
- Less prone to overfitting (constrained depth).
- Computationally efficient (far fewer evaluations than exhaustive search).

These constrained parameters confirm our hypothesis that automated optimization can produce models that are both accurate and explainable. As shown in Table 10, the Hyperopt-tuned DT (H-DT) achieves a slightly higher

G-means (0.8945 vs. 0.8875) and recall (0.8047 vs. 0.7912) than the DT obtained by exhaustive search (C-DT), while reducing execution time by 80% (54.65 vs. 273.64 min), although C-DT retains a higher precision and PR AUC. The false positives produced by the tuned models are partly a consequence of SMOTE over-sampling [32], and since they represent an operational workload, they are analysed in detail in Section 5.4.1.

The parameter effects give insight into the impact of different hyperparameter combinations on the evaluation metric. They also indicate which parameters have the most influence on G-mean, in this case, the depth and the minimum information gain have the greatest effect on the model's performance.

The convergence illustrated in Figure 2 demonstrates that even when the evaluation metric in a given iteration yields a poor result, the meta-heuristic is still capable of returning to a good parameter combination that leads to a well-performing model. It is important to recall that the final hyperparameter combination is determined by the best evaluation metric obtained during the optimization process. The evaluations with a G-means of zero (Figures. 2 and 3) correspond to configurations that predict no fraud at all (recall = 0), which shows how sensitive single trees and forests are to their hyperparameters on imbalanced data.

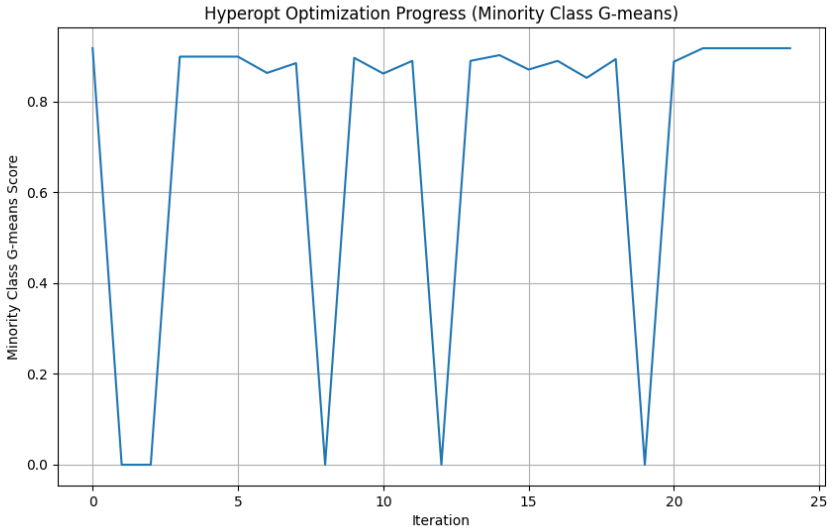


Figure 2. Convergence of G-means for Hyperop DT (H-DT).

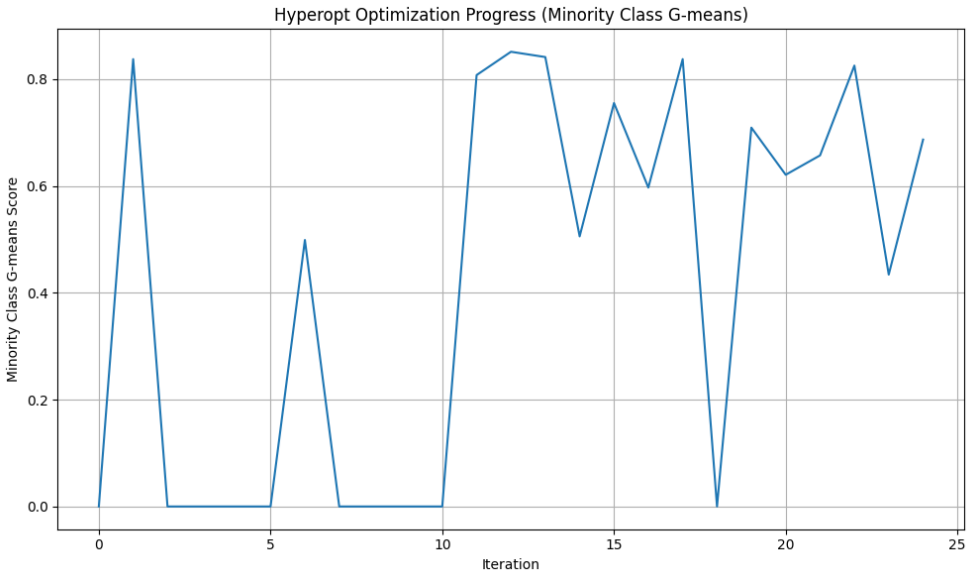


Figure 3. Convergence of G-means for H-RF.

For the RF algorithm, the full combination execution failed due to large tasks related to Spark configuration. This error cannot be avoided, as our current configuration is the best possible given our hardware resources. Therefore, for this algorithm, we can strongly confirm that the model trained using the meta-heuristic approach is more effective in big data environments. The model demonstrated strong predictive performance and solid overall KPIs, especially for imbalanced data (Precision, G-mean, Recall and PRAUC), making it the best overall model. The convergence of H-RF (Figure 3) leads to the same interpretation as for the Decision Tree and further confirms that the heuristic is robust in returning to good parameter combinations after poor iterations. They also highlight the crucial importance of hyperparameter tuning: the number of false positives (Type I error, or α error) has significantly decreased (from about 4,889 for S-RF to about 445 for H-RF, Table 11), which is a very positive sign when dealing with highly imbalanced data.

The GBT model also reaches a good G-means value (Figure 4, all evaluations between about 0.79 and 0.90), but it produces a high number of false positives, which is a major drawback in high-risk domains such as fraud detection or credit risk.

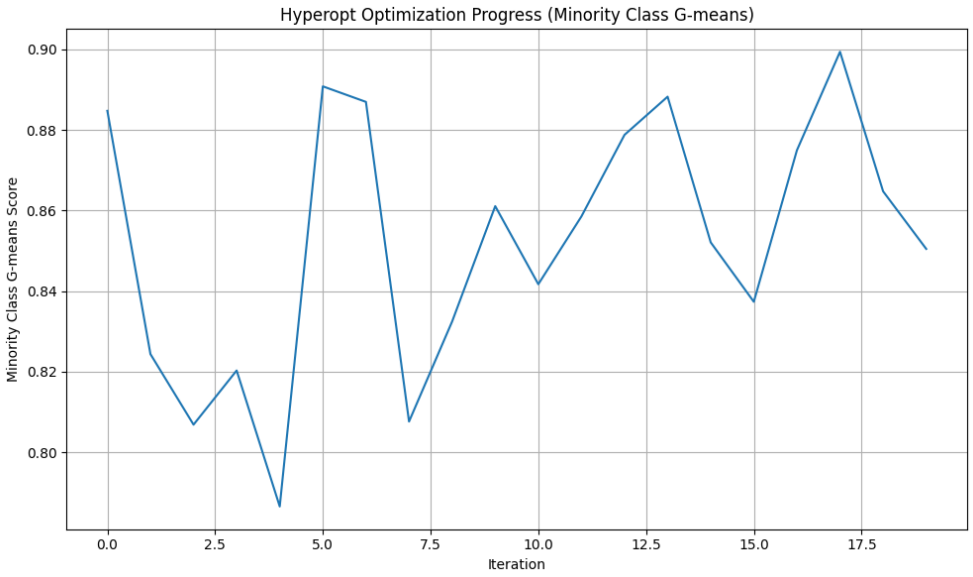


Figure 4. Convergence of G-means for H-GBT.

5.3. Key performance indicators (KPIs)

5.3.1. Quantitative KPIs:

Table 10 reports the performance metrics of the trained models, and the Hyperopt-tuned models were obtained with 25 TPE evaluations (20 for GBT). The results in Table 10 demonstrate compelling performance for models using metaheuristic optimization, with significantly reduced execution times compared to complex models (where execution completed).

5.3.2. Qualitative KPIs:

Our fraud detection system's performance is monitored through several key metrics, particularly focusing on processing latency. The system demonstrates variable response times ranging from 3.5 to 5 seconds for complete transaction processing from initial data streaming through prediction generation to final storage in HBase. These latency measurements, which represent the time elapsed between streaming initiation and final results, are directly influenced by available computational resources. We observe that processing times tend to increase proportionally with the duration of system operation, suggesting that resource allocation and optimization become increasingly important for maintaining consistent performance over extended periods. The measured latencies of $4s$, $3.5s$ and $5s$ provide concrete benchmarks for evaluating the system's real-time processing capabilities in operational environments. They include micro-batch scheduling, feature transformation, model inference and the HBase write. A mean of about 4.2 s is compatible with near-real-time monitoring and post-authorisation alerting, but not with in-line blocking within a sub-second authorisation window. For reference, Daksa and Kemala [16] report 0.8–0.9 s with Spark at 10–40 transactions per second, under different ingestion rates, pipeline components and hardware, so a direct comparison is not possible.

5.3.3. Resource consumption:

Table 9 reports the resource consumption measured during the processing of our data set (1.52 GiB), together with the same values normalised per GiB of input data. CPU and memory requirements are driven mainly by the Spark processing jobs, the Kafka producer and consumer instances and the visualisation processes of the monitoring dashboard. The normalised values provide a basis for capacity planning, while a systematic throughput and scalability benchmark across larger data volumes and cluster sizes is left for future work (Section 5.5).

Table 9. Resource consumption measured on our data set and normalised per GiB of input.

	CPU (%)	Number of CPU Cores	RAM (GiB)
Data set (1.52 GiB)	83.7	0.84	1.37
Per GiB of input	55.1	0.55	0.90

5.4. Performance evaluation

Quantitative comparison of the proposed model against baseline algorithms.

- For imbalanced datasets, prioritize G-means and PR AUC to assess minority-class detection.
- H-RF leads in PR AUC (0.6123), surpassing S-GBT (0.4710), H-GBT (0.4479), H-DT (0.2604) and others.
- It balances recall (0.7811), precision (0.3427) and specificity (0.9978) with G-means 0.8828.
- Its runtime (118.59 min) is shorter than that of C-DT (273.64 min) but longer than that of H-DT (54.65 min).

Comparative model performance research (Table 10) shows clear trends among various algorithm kinds and implementations. DT variants (S-DT, C-DT and H-DT) show a distinct trade-off between predictive strength and computing efficiency, with execution times varying from 273.64 minutes for the exhaustive grid search (C-DT) to barely 1 minute for the standard version. Given its good metrics (G-means, Recall) and manageable execution time (54.65 minutes), the Hyperopt-tuned DT (H-DT) stands out as a particularly well-balanced approach. RF models exhibit even more pronounced differences: the Hyperopt-tuned version (H-RF) attains the best precision, F1, MCC and PR AUC of all examined models, whereas the standard implementation S-RF struggles with precision despite moderate recall.

An interesting case study in performance optimization is provided by the GBT implementations. The execution timings of the GBT variants range greatly from 7.74 minutes (S-GBT) to 194.54 minutes (H-GBT), despite the fact that all of them exhibit competitive recall values (0.7475-0.8047). Among all setups, the Hyperopt-tuned GBT model has the highest G-means score (0.8946, practically equal to H-DT), even at a significant computational cost. When implementing fraud detection systems in production settings, machine learning practitioners must manage the fundamental conflict between model complexity and operational efficiency, as this pattern illustrates.

A number of significant findings are drawn from the full assessment. First, Hyperopt-tuned models outperform their standard counterparts on most metrics, and match or exceed the exhaustive grid search where it could be completed (DT) at a fraction of its cost.

Table 10. ML Models Performance Metrics (Hyperopt: 25 TPE evaluations for DT and RF, 20 for GBT)

	S-DT	C-DT	H-DT	S-RF	C-RF	H-RF	S-GBT	C-GBT	H-GBT
Precision	0.0911	0.2087	0.1755	0.041	-	0.3427	0.2042	*	0.1646
Recall	0.6195	0.7912	0.8047	0.7037	-	0.7811	0.7475	*	0.8047
F1	0.1589	0.3303	0.2881	0.0776	-	0.4764	0.3208	*	0.2733
Specificity	0.9908	0.9955	0.9944	0.9756	-	0.9978	0.9957	*	0.9939
B-Accuracy	0.8052	0.8934	0.8995	0.8396	-	0.8895	0.8716	*	0.8993
MCC	0.2351	0.4049	0.3742	0.1661	-	0.5164	0.3892	*	0.3622
ROC AUC	0.9368	0.8934	0.9475	0.938	-	0.9651	0.9713	*	0.9418
G-means	0.7835	0.8875	0.8945	0.8226	-	0.8828	0.8627	*	0.8946
PR AUC	0.1511	0.4998	0.2604	0.3026	-	0.6123	0.471	*	0.4479
T-execution	1 min	273.64 min	54.65 min	3 min	-	118.59 min	7.74 min	*	194.54 min

-: Execution error due to a loop running for more than 400 minutes.
*: Execution time exceeds 550 minutes.
All models are evaluated on the same test set, which contains 297 actual fraudulent transactions (support of the fraud class).

Second, performance measurements for the RF architecture fluctuate more than those for other model types, indicating that it is the most sensitive to implementation decisions. The highest-performing models (especially H-GBT and H-RF) demand noticeably more processing time, demonstrating the trade-off between model sophistication and computational resources.

5.4.1. False positives and operational workload.

In banking, a false positive leads to an analyst review or to a declined legitimate payment, so the number of false alarms is an operational constraint [23]. Table 11 derives the approximate confusion counts from the precision, recall and specificity of Table 10 and the 297 test frauds. Precision (0.3427 for H-RF) should not be confused with the false-positive rate: the false-positive rate of H-RF is $1 - 0.9978 = 0.22\%$ of legitimate transactions, i.e. about 445 false alarms for about 232 detected frauds (1.9 false alarms per detected fraud, or about 22 alerts per 10,000 legitimate transactions).

Table 11. Approximate test-set confusion counts derived from the metrics of Table 10 (297 frauds, $\approx 2.0 \times 10^5$ legitimate transactions, rounded values).

Model	TP	FN	FP	FP rate	FP per TP
S-DT	184	113	1,836	0.92%	10.0
C-DT	235	62	891	0.45%	3.8
H-DT	239	58	1,123	0.56%	4.7
S-RF	209	88	4,889	2.44%	23.4
H-RF	**232**	**65**	**445**	**0.22%**	**1.9**
S-GBT	222	75	865	0.43%	3.9
H-GBT	239	58	1,213	0.61%	5.1

Two factors explain the remaining false positives. First, SMOTE over-sampling together with the G-means objective moves the decision boundary towards the fraud class, favouring recall at the cost of precision [30, 32], and for GBT, TPE raises recall from 0.748 to 0.805 but also raises false positives from about 865 to about 1,213. Second, all models use the default threshold of 0.5, although resampling distorts the predicted probabilities [30]. This level of false alarms is not acceptable as such in production, where the number of alerts must match the review capacity of the fraud team. Three levers can reduce it: (i) calibrating the scores and selecting the decision threshold according to the alert budget or to the relative costs of false negatives and false positives [1, 23], (ii) optimising a cost-sensitive or precision-constrained objective instead of G-means, and (iii) using the model output as a risk score that prioritises analyst review rather than automatically blocking transactions. Among the evaluated models, H-RF is the most suitable starting point, with about 91% fewer false alarms than S-RF and 50% fewer than C-DT at a recall of 0.78.

5.5. Discussion

The results of Sections 5.1–5.4 answer the research question in a nuanced way.

Hybrid resampling combined with TPE tuning yields interpretable models that detect about 78–80% of frauds while flagging 0.2–0.6% of legitimate transactions (Tables 10 and 11). H-RF provides the best balance between detection and false alarms (PR AUC 0.612, MCC 0.516), while H-DT and H-GBT reach a slightly higher recall and G-means but with 2.5 to 2.7 times more false positives. The residual false-alarm level of H-RF (about 1.9 alerts per detected fraud) remains significant and must be reduced by threshold calibration or cost-sensitive optimisation before production use (Section 5.4.1).

Exhaustive grid search is not viable for tree ensembles on a resource-constrained Spark deployment: it failed for RF and GBT, whereas TPE produced tuned models in 118.59 and 194.54 min. For DT, TPE obtained a comparable G-means with 80% less time (Sections 5.1–5.2). This confirms offline observations [10, 35, 36] in a distributed fraud detection pipeline. The choice of objective matters: optimising G-means favours recall, while an objective aligned with operational costs would favour precision.

The decoupled Lambda architecture makes it possible to retrain and redeploy models without interrupting ingestion, and Kafka partitioning allows the speed layer to be parallelised. The measured end-to-end latency of 3.5–5 s (Section 5.3) is adequate for near-real-time alerting, but not yet for in-line authorisation decisions.

Our predictive results are in line with those reported for tree-based ensembles on real card data [23]. Many studies on the public European data set report accuracies above 98% [9, 10], but these are not directly comparable: accuracy is dominated by the majority class, whereas our evaluation uses a chronological split, a stronger imbalance (675:1 versus about 576:1) and imbalance-aware metrics. The added value of this study is therefore an end-to-end, reproducible pipeline evaluated on real banking data, with an explicit account of tuning cost, false alarms and latency, rather than a new state-of-the-art score.

5.5.1. Limitations.

(i) The data set is confidential, which prevents public replication, although the pipeline relies only on open-source components. (ii) The evaluation uses a single chronological split, without repeated runs or confidence intervals. (iii) Only tree-based models were evaluated, and SVMs and neural networks [9, 29] would provide additional baselines, at the cost of interpretability and, for SVMs, of training time at scale. (iv) The decision threshold was not optimised and no cost-sensitive objective was used. (v) Concept drift and verification latency are not explicitly modelled. (vi) Latency was measured on a few runs, and scalability has not yet been benchmarked beyond the volume of our data set.

6. Conclusion

We developed a system for detecting financial fraud in real time based on Apache Kafka, Spark and MLlib, organised in a containerised Lambda architecture. On confidential card data from a Moroccan bank with a 675:1 imbalance, hybrid resampling (random under-sampling and SMOTE) combined with TPE-based tuning produced interpretable tree-based models. The TPE-tuned Random Forest achieved the best trade-off (PR AUC 0.612, recall 0.781, specificity 0.998) and about 91% fewer false alarms than its default configuration. TPE reduced tuning time by 80% relative to grid search for Decision Trees and made the tuning of Random Forest and Gradient-Boosted Trees feasible where grid search failed. End-to-end latency was 3.5 to 5 seconds. We also observed that DataOps practices (containerisation and version-controlled deployment) helped to automate the system.

Future work will address the limitations identified in Section 5.5: (i) calibrating the decision threshold and optimising a cost-sensitive objective to match the alert budget of fraud analysts, (ii) adding SVM and neural network baselines, including SVMs with column generation for feature selection, (iii) incremental retraining and drift detection to follow evolving fraud patterns [26], (iv) a systematic throughput, latency and scalability benchmark across data volumes and cluster sizes, with auto-scaling of resources, and (v) assessing the fairness and explainability of the decisions [27].

Acknowledgments

The authors thank the editor and the anonymous reviewers for their careful reading of the manuscript and for their constructive comments and suggestions, which substantially improved the quality of this work.

Funding, Competing Interests, and Data Availability

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Competing interests: All authors certify that they have no affiliations with or involvement in any organization or entity with any financial or non-financial interest in the subject matter or materials discussed in this manuscript.

Data availability: The transaction data used in this study were provided by a Moroccan bank under a non-disclosure agreement and cannot be made publicly available.

Author Contributions

R.H.: Conceptualization, Methodology, Software, Investigation, Data curation, Formal analysis, Writing: original draft, Writing: review and editing.

Y.T.: Software, Investigation, Data curation.

M.O.: Conceptualization, Methodology, Supervision, Writing: review and editing.

K.E.M.: Supervision, Writing: review and editing.

All authors contributed to the validation of the results and approved the final version of the manuscript.

References

[1] R. Hakimi, B. Benyacoub, M. Ouzineb, Unveiling the power of cost sensitivity: A comparative study of different models on imbalanced datasets in machine learning, in: Communication and Information Technologies through the Lens of Innovation, Springer Nature Switzerland, 2025, pp. 113–120. doi:10.1007/978-3-031-74470-9_14.

[2] T. Watthaisong, K. Sunat, N. Muangkote, Comparative evaluation of imbalanced data management techniques for solving classification problems on imbalanced datasets, Statistics, Optimization & Information Computing 12 (2) (2024) 547–570. doi:10.19139/soic-2310-5070-1890.

[3] E. Osegi, E. Jumbo, Comparative analysis of credit card fraud detection in simulated annealing trained artificial neural network and hierarchical temporal memory, Machine Learning with Applications 6 (2021) 100080. doi:10.1016/j.mlwa.2021.100080.

[4] A. Izotova, A. Valiullin, Comparison of poisson process and machine learning algorithms approach for credit card fraud detection, Procedia Computer Science 186 (2021) 721–726. doi:10.1016/j.procs.2021.04.214.

[5] J. Domashova, O. Zabelina, Detection of fraudulent transactions using sas viya machine learning algorithms, Procedia Computer Science 190 (2021) 204–209. doi:10.1016/j.procs.2021.06.025.

[6] G. S. Thejas, S. Dheeshjith, S. Iyengar, N. Sunitha, P. Badrinath, A hybrid and effective learning approach for click fraud detection, Machine Learning with Applications 3 (2021) 100016. doi:10.1016/j.mlwa.2020.100016.

[7] Y. Shen, C. Guo, H. Li, J. Chen, Y. Guo, X. Qiu, Financial feature embedding with knowledge representation learning for financial statement fraud detection, Procedia Computer Science 187 (2021) 420–425. doi:10.1016/j.procs.2021.04.110.

[8] A. Cherif, A. Badhib, H. Ammar, S. Alshehri, M. Kalkatawi, A. Imine, Credit card fraud detection in the era of disruptive technologies: A systematic review, Journal of King Saud University - Computer and Information Sciences 35 (1) (2023) 145–174. doi:10.1016/j.jksuci.2022.11.008.

[9] F. K. Alarfaj, I. Malik, H. U. Khan, N. Almusallam, M. Ramzan, M. Ahmed, Credit card fraud detection using state-of-the-art machine learning and deep learning algorithms, IEEE Access 10 (2022) 39700–39715. doi:10.1109/access.2022.3166891.

[10] A. A. Taha, S. J. Malebary, An intelligent approach to credit card fraud detection using an optimized light gradient boosting machine, IEEE Access 8 (2020) 25579–25587. doi:10.1109/access.2020.2971354.

[11] C. Martín, P. Langendoerfer, P. S. Zarrin, M. Díaz, B. Rubio, Kafka-ml: Connecting the data stream with ml/ai frameworks, Future Generation Computer Systems 126 (2022) 15–33. doi:10.1016/j.future.2021.07.037.

[12] D. R. Torres, C. Martín, B. Rubio, M. Díaz, An open source framework based on kafka-ml for distributed dnn inference over the cloud-to-things continuum, Journal of Systems Architecture 118 (2021) 102214. doi:10.1016/j.sysarc.2021.102214.

[13] E. Nagy, R. Lovas, I. Pintye, Á. Hajnal, P. Kacsuk, Cloud-agnostic architectures for machine learning based on apache spark, Advances in Engineering Software 159 (2021) 103029. doi:10.1016/j.advengsoft.2021.103029.

[14] A. Benítez-Hidalgo, C. Barba-González, J. García-Nieto, P. Gutiérrez-Moncayo, M. Paneque, A. J. Nebro, M. d. M. Roldán-García, J. F. Aldana-Montes, I. Navas-Delgado, Titan: A knowledge-based platform for big data workflow management, Knowledge-Based Systems 232 (2021) 107489. doi:10.1016/j.knosys.2021.107489.

[15] F. Carcillo, A. Dal Pozzolo, Y.-A. Le Borgne, O. Caelen, Y. Mazzer, G. Bontempi, SCARFF : A scalable framework for streaming credit card fraud detection with spark, Information Fusion 41 (2018) 182–194. doi:10.1016/j.inffus.2017.09.005.

[16] R. P. Daksa, A. P. Kemala, A comparative study on real time data streaming for fraud detection using kafka with apache flink and apache spark, Procedia Computer Science 269 (2025) 192–199. doi:10.1016/j.procs.2025.08.272.

[17] M. S. ElSayed, N.-A. Le-Khac, M. A. Albahar, A. Jurcut, A novel hybrid model for intrusion detection systems in sdns based on cnn and a new regularization technique, Journal of Network and Computer Applications 191 (2021) 103160. doi:10.1016/j.jnca.2021.103160.

[18] G. Suchacka, A. Cabri, S. Rovetta, F. Masulli, Efficient on-the-fly web bot detection, Knowledge-Based Systems 223 (2021) 107074. doi:10.1016/j.knosys.2021.107074.

[19] M. de Vos, J. Pouwelse, Contrib: Maintaining fairness in decentralized big tech alternatives by accounting work, Computer Networks 192 (2021) 108081. doi:10.1016/j.comnet.2021.108081.

[20] Y. Luo, X. Zou, D. Liu, W. Peng, X. Wu, Extraction of specific arguments from chinese financial news with out-of-domain samples, Procedia Computer Science 183 (2021) 288–294. doi:10.1016/j.procs.2021.02.061.

[21] D. Malhotra, A. Chug, A modified label propagation algorithm for community detection in attributed networks, International Journal of Information Management Data Insights 1 (2) (2021) 100030. doi:10.1016/j.jjimei.2021.100030.

[22] W. Hilal, S. A. Gadsden, J. Yawney, Financial fraud: A review of anomaly detection techniques and recent advances, Expert Systems with Applications 193 (2022) 116429. doi:10.1016/j.eswa.2021.116429.

[23] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, G. Bontempi, Credit card fraud detection: A realistic modeling and a novel learning strategy, IEEE Transactions on Neural Networks and Learning Systems 29 (8) (2018) 3784–3797. doi:10.1109/tnnls.2017.2736643.

[24] H. He, E. Garcia, Learning from imbalanced data, IEEE Transactions on Knowledge and Data Engineering 21 (9) (2009) 1263–1284. doi:10.1109/tkde.2008.239.

[25] K. Niu, Z. Zhang, Y. Liu, R. Li, Resampling ensemble model based on data distribution for imbalanced credit risk evaluation in p2p lending, Information Sciences 536 (2020) 120–134. doi:10.1016/j.ins.2020.05.040.

[26] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys 46 (4) (2014) 1–37. doi:10.1145/2523813.

[27] I. Psychoula, A. Gutmann, P. Mainali, S. H. Lee, P. Dunphy, F. Petitcolas, Explainable machine learning for fraud detection, Computer 54 (10) (2021) 49–59. doi:10.1109/mc.2021.3081249.

[28] S. Lessmann, B. Baesens, H.-V. Seow, L. C. Thomas, Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research, European Journal of Operational Research 247 (1) (2015) 124–136. doi:10.1016/j.ejor.2015.05.030.

[29] I. D. Mienye, N. Jere, Deep learning for credit card fraud detection: A review of algorithms, challenges, and solutions, IEEE Access 12 (2024) 96893–96910. doi:10.1109/access.2024.3426955.

[30] A. Dal Pozzolo, O. Caelen, R. A. Johnson, G. Bontempi, Calibrating probability with undersampling for unbalanced classification, in: 2015 IEEE Symposium Series on Computational Intelligence, IEEE, 2015, pp. 159–166. doi:10.1109/ssci.2015.33.

[31] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, Smote: Synthetic minority over-sampling technique, Journal of Artificial Intelligence Research 16 (2002) 321–357. doi:10.1613/jair.953.

[32] A. Fernandez, S. Garcia, F. Herrera, N. V. Chawla, Smote for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary, Journal of Artificial Intelligence Research 61 (2018) 863–905. doi:10.1613/jair.1.11192.

[33] T. Saito, M. Rehmsmeier, The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets, PLOS ONE 10 (3) (2015) e0118432. doi:10.1371/journal.pone.0118432.

[34] D. Chicco, G. Jurman, The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation, BMC Genomics 21 (1) (2020). doi:10.1186/s12864-019-6413-7.

[35] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of Machine Learning Research 13 (2012) 281–305.

[36] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: Advances in Neural Information Processing Systems 24 (NIPS 2011), 2011, pp. 2546–2554.

[37] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, Apache spark: a unified engine for big data processing, Communications of the ACM 59 (11) (2016) 56–65. doi:10.1145/2934664.