# Fraud Detection for Financial Transactions - A Solution Design Specification

INT3229E_2: Big Data Techniques and Technologies – Team 2-4

Nguyen Thanh Dao[a], Nguyen Van Thien[a], Nguyen Huu Phuong[a], Pham Ngoc Linh[a], Tran Thu Thuy[a]

[a] *University of Engineering and Technology, VNU, Hanoi, Vietnam*

**Abstract**

This report presents the design and implementation of a scalable, real-time fraud detection system for financial transactions. It leverages modern Big Data technologies to overcome the limitations of traditional rule-based methods. The system architecture incorporates Apache Kafka for high-throughput data ingestion, Apache Spark and Apache Flink for distributed stream processing, Google BigQuery for scalable storage, and Grafana for real-time visualization. Machine learning models, trained using H2O Sparkling Water - an AutoML framework, are integrated to detect fraudulent behaviors. Model performance is evaluated using standard metrics such as precision, recall, F1-score, and ROC AUC, with special attention to the challenges posed by class imbalance. By using both empirical and synthetic datasets, the system ensures robustness and reflect real-world fraud patterns.

The solution also includes an interactive dashboard that supports real-time monitoring and investigation of fraud events. Experimental results demonstrate that Spark is well-suited for high-throughput scenarios, whereas Flink offers advantages in low-latency environments. Overall, the system provides a flexible and efficient framework for real-time fraud detection in dynamic and data-intensive financial ecosystems.

*Keywords:* Fraud, Financial Transaction, Spark, Kafka, BigQuery, Grafana

## 1. Introduction

Conventional fraud detection systems primarily rely on static, rule-based approaches that depend on predefined patterns or thresholds. While these systems can be effective for well-known types of fraud, they are rigid and often incapable of identifying new or subtle fraudulent behaviors [1]. Additionally, these approaches typically operate in batch mode, which introduces latency and prevents real-time response to fast-moving threats.

These limitations are increasingly exposed in today's financial landscape, where thousands of transactions may occur per second. Modern fraud scenarios require dynamic detection systems that can learn from data streams and adapt to new patterns in real time [2]. Furthermore, the rise of fraud techniques such as synthetic identities, bot-driven attacks, and cross-platform fraud highlights the limitations of rule-based engines and underscores the need for machine learning-driven, adaptive detection methods [3].

This project aims to design and implement a scalable data processing pipeline that can efficiently detect fraudulent activities in real-time. By leveraging a suite of big data technologies, including Apache Kafka for data ingestion, Apache Spark for stream processing, Google BigQuery for distributed storage, and Grafana for monitoring and visualization, we aim to build a robust system capable of handling large-scale transaction data.

This project addresses these challenges by designing a scalable, real-time fraud detection pipeline built on Big Data technologies, enabling continuous monitoring and adaptive learning for improved fraud resilience.

The primary objectives of this project are:

- To design a real-time data processing architecture that can ingest, process, and analyze financial transaction data at scale.
- To apply anomaly detection techniques and machine learning models to identify potentially fraudulent activities.
- To ensure system reliability, scalability, and low-latency response times suitable for real-world financial environments.
- To visualize system metrics and fraud detection results through an interactive dashboard for monitoring and evaluation.

## 2. Functional Requirements

The Fraud Detection System is designed to provide a set of core functionalities that enable effective, scalable, and real-time analysis of financial transactions. The key features are as follows:

- **Real-Time Data Ingestion:** The system should continuously ingest transaction data from various sources using Apache Kafka.

- **Fraud Detection and Alerting:** The system should apply a machine learning model for detecting fraudulent transactions in real-time using Apache Spark and immediately raise alerts for suspicious transactions.
- **Dashboard and Visualization:** System metrics, fraud trends, and alerts should be visualized in an intuitive dashboard using Grafana.
- **Data Storage and Retrieval:** All data—including raw transaction records, feature-engineered data, detection results, and alert logs should be persistently stored using BigQuery. The storage system must support historical querying and analysis, and facilitate model retraining by providing access to labeled and time-stamped datasets.
- **Model Training and Updating:** The system should support updating of machine learning models using labeled data.

## 3. Non-Functional Requirements

In addition to core functionalities, the Fraud Detection System must satisfy the following non-functional requirements to ensure high performance, security, and maintainability in a production-grade big data environment:

- **Performance:**
  - Throughput: The system must be capable of processing up to 100 transactions per second (TPS).
  - The latency between data ingestion and fraud alert generation should not exceed 10 seconds under normal operating conditions.
- **Scalability:** The system architecture should support horizontal scaling to accommodate varying transaction volumes. All components (Kafka, Spark, storage) should be deployable on distributed clusters and scalable independently. As the system scales, key metrics such as latency and throughput should remain within acceptable bounds.

## 4. System Architecture & Implementation Plan

The proposed Fraud Detection System is designed as a real-time data processing pipeline that integrates various Big Data technologies including Kafka, Apache Spark, Apache Flink, BigQuery, and Grafana. The system enables ingestion, transformation, prediction, storage, and visualization of transaction data for the purpose of detecting fraudulent activity at scale. Figure 1 presents the high-level architecture of the serving system consisting of 4 main modules: Data Ingestion, Data Processing, Data Storage, and Visualization. The details of the architecture and implementation plans of these modules are presented in the following subsections.

### *4.1. Architecture*

#### *4.1.1. Data Ingestion*

In the Data Ingestion module, we use Kafka to ingest data. The purpose of this module is to consume data from various sources and output a "Transaction" topic that stores unified data, waiting to be transferred to further processing steps.

- Step 1: Multiple data sources (Producer 1, 2, and 3) stream transaction data into Apache Kafka. Kafka acts as a central message broker, aggregating the data.
- Step 2: Brokers of Kafka preprocess the original data and convert them into a standard format and push the data to a logical data stream called Transaction topic, where later processing layer can pull data from.
- Step 3: After messages have been pushed to the Transaction topic, the standardized data will be stored in BigQuery. Here, the data stream is gathered by a *Connector* (powered by CONFLUENT) before being stored in Big Query.

#### *4.1.2. Data Processing*

In the Data Processing module, we use Apache Spark and Apache Flink to process transaction records in a real-time manner. The reason for the utilization of both Spark and Flink is for the purpose of serving different data rate scenarios. Spark is best suitable for cases where we need to process data batches, which mean the data coming from producers is rather high. Meanwhile, in cases where the data rate is low, using Flink can help with decreasing latency of the system. In both Spark and Flink, we integrate feature engineering and ML prediction models for detecting frauds. To be more specific:

- Step 4: Transaction topic of Kafka can be considered as a data stream. The data stream needs to be processed to be valuable and insightful. Therefore, we use Spark and Flink to consume data from Transaction topic to apply complex processing.
- Step 5: After being streamed to Spark and Flink, the data goes through a layer of Feature Engineering and Transformation which makes the data more suitable for prediction model.
- Step 6: The data after being engineered will be fed to a machine learning model for predicting fraud transaction. Here, the model we use is trained by H2O.ai, a AutoML framework that automates the processes of training, testing ML models.

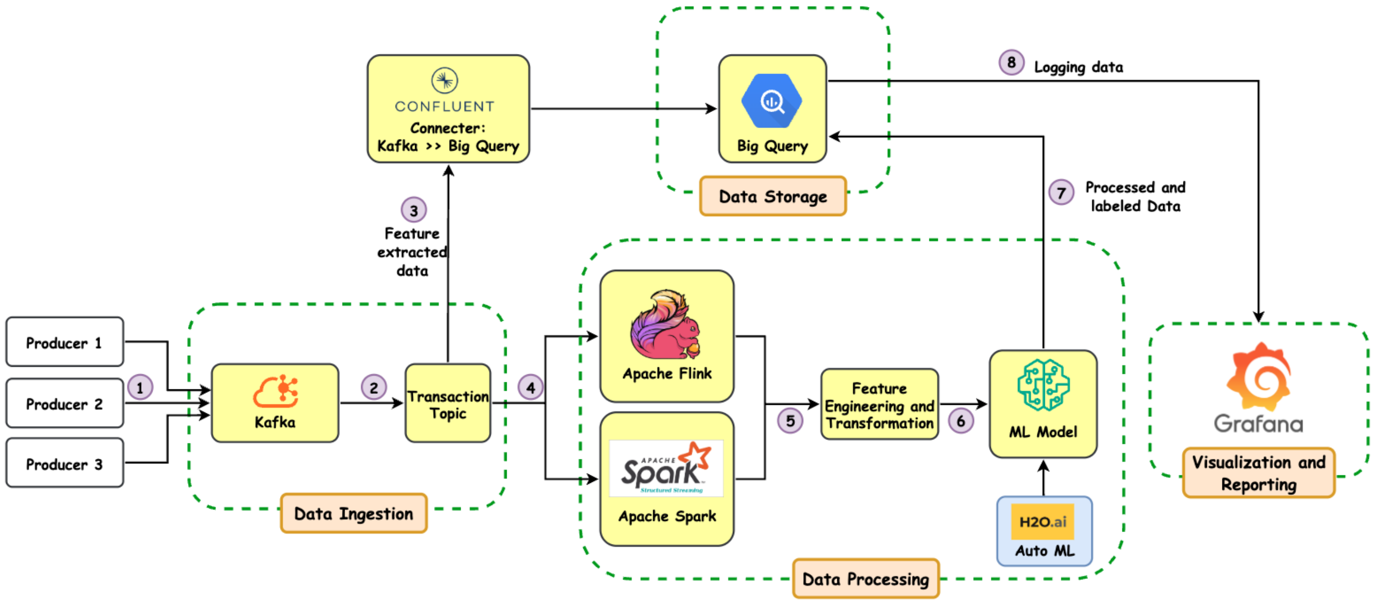

Figure 1: System Architecture of the Fraud Detection Platform

### *4.1.3. Data Storage*

Google BigQuery is employed in our system to provide scalable, fault-tolerant, and distributed storage for large volumes of transaction data. It ensures persistent storage of transaction data before and after being processed, supporting long-time data storage and historical analysis. Below, we describe the steps involved with BigQuery storage:

- Step 3: Mentioned earlier in Data Ingestion module. In this step, data consumed from Kafka is stored to BigQuery. Connector decode the encrypted data stored in Kafka and store it to BigQuery.
- Step 7: Results returned from the ML model (including transaction id, and label, processed time) will be stored in BigQuery. These data will be used for monitoring throughput, latency and fraud count of the system.

### *4.1.4. Visualization*

In this project, we use Grafana to monitor and visualize real-time system metrics such as processing throughput, end-to-end latency, geographical statistics, and fraud amount distributions. This setup provides intuitive dashboards, allowing administrators to detect anomalies and respond promptly.

- Step 8: Historical data of fraud prediction will be utilized to visualize various informative figures, such as: processing throughput, end-to-end latency, geographical statistics, fraud amount distributions. (More details in Section VII)

## *4.2. Implementation*

Our project heavily relies on Google Cloud services including Compute Engine, BigQuery, and Cloud Storage for deploying system components. Additionally, our Grafana server is hosted using the Grafana Cloud service. Further details are provided below.

### *4.2.1. Data Ingestion*

To simulate real-time data streams, Python-based producer applications are run on VM instances on Google Cloud, mimicking the behavior of transaction sources in a real-world scenario. In this project, we experiment using three producers, which have configurations and capacities presented in Table 1

Table 1: Transactions Producers' Deployments on GCE

| VM Name | OS | RAM | MEM | Storage | Max Data Rate |
|---|---|---|---|---|---|
| producer-1 | Debian | 4GB | 2 vCPUs | 10GB | 100 TPS |
| producer-2 | Debian | 4GB | 2 vCPUs | 10GB | 100 TPS |
| producer-3 | Debian | 4GB | 2 vCPUs | 10GB | 100 TPS |

The Kafka-based data ingestion module consists of a cluster of brokers and a manager that collaboratively consume messages from producers. In our project, this Kafka layer is deployed using Compute Engine instances, as outlined in Table 2.

Table 2: Kafka Cluster Deployment on GCE

| VM Name | Role | OS | RAM | MEM | Storage |
|---|---|---|---|---|---|
| kafka-broker-1 | Broker 1 | Debian | 4GB | 2 vCPUs | 10GB |
| kafka-broker-2 | Broker 2 | Debian | 4GB | 2 vCPUs | 10GB |
| kafka-broker-3 | Broker 3 | Debian | 4GB | 2 vCPUs | 10GB |
| zookeeper | Zookeeper | Debian | 4GB | 2 vCPU | 10GB |

*4.2.2. Data Processing*

The Spark-based data processing layer also operates on Compute Engine VMs of Google Cloud. For the ease of implementation, we created a Docker image for the Spark instances [1]. Because of the containerization, our Spark instances can be scaled very easily. Here, we create a cluster of a total of 3 Spark workers and 1 master for the Data Processing module. The configurations of these instances are outlined in Table 3

Table 3: Spark Cluster Deployment (Docker containers) on GCE

| VM Name | Mode | OS | RAM | MEM | Storage |
|---|---|---|---|---|---|
| worker-1 | Worker | Debian | 2.8GB | 2 vCPUs | 5GB |
| worker-2 | Worker | Debian | 2.8GB | 2 vCPUs | 5GB |
| worker-3 | Worker | Debian | 2.8GB | 2 vCPUs | 5GB |
| master | Master | Debian | 2.8GB | 2 vCPU | 5GB |

**Note:** Since the Spark instances are deployed within containers, their performance may be reduced compared to running directly on the host operating system. To maximize performance, users are encouraged to run the code directly from the GitHub repository rather than using Docker containers.

*4.2.3. Data Storage*

In our project, we store the data processed by the Data Processing layer in Google BigQuery. BigQuery offers a fully managed, scalable, and serverless data warehouse solution that is well-suited for handling large volumes of structured streaming data. This integration allows efficient querying and analysis of processed data without the need to manage a separate storage system.

In our fraud detection system, it is essential to continuously send transactional data from Kafka to BigQuery for analysis, reporting, and long-term data retention. To automate this process, we use the BigQuery Sink Connector, which consumes messages from a Kafka topic and writes them directly into a BigQuery table without manual intervention or scheduled batch jobs.

Kafka connectors are plugins used within the Kafka Connect framework to facilitate the movement of large data sets between Apache Kafka and external data systems. They enable seamless integration by abstracting the technical details of connecting to various data sources or sinks, allowing users to build complex data pipelines without writing custom code. [4]

The BigQuery Sink Connector connector operates in near real-time, ensuring that transaction records are quickly available in BigQuery as soon as they are produced in Kafka. This rapid flow of data is crucial for detecting and responding to fraud in a timely manner. In addition to moving data, the connector can automatically create tables if they do not already exist, handle message batching for performance, and serialize data using Avro to maintain structure and compatibility with downstream tools. [5]

[1]`https://github.com/png261/spark-kafka-bigquery`

*4.2.4. Visualization*

For the visualization layer of our fraud detection system, we will employ Grafana, a powerful open-source data visualization and monitoring platform. Grafana will be hosted on Grafana Cloud, allowing scalable and accessible visualization. To enable real-time monitoring and insightful analysis, Grafana will be configured to interact with data collected by Prometheus, which gathers metrics generated by our processing pipeline.

## 5. Data Design

*5.1. Datasets Criteria*

The main objective of this project is to design and implement a scalable system capable of detecting fraudulent financial transactions using machine learning techniques. Given the project's Big Data context, the chosen dataset must meet the following criteria:

- **Large-scale:** Provide a realistic BigData environment to test the capabilities of distributed processing tools such as Kafka, Flink or Spark.
- **Class imbalance:** Represents the typical imbalance in fraud detection problems where fraudulent transactions are rare.
- **Benchmark-friendly:** Recognized and widely used for model comparison and benchmarking in the research community.
- **Synthetic expandability:** Enables data generation for testing, experimentation, and improving model robustness.

*5.2. Datasets Overview*

In building a robust fraud detection system, it is essential to train and evaluate models using diverse data sources. For this project, we use two datasets: (i) empirical (the ULB Credit Card Fraud Detection dataset)[6], and (ii) synthetic dataset (the Kartik Fraud Detection dataset)[7]. This dual approach enhances the model's learning capacity and generalization abilities, especially when dealing with rare and evolving fraud patterns.

The empirical dataset is based on real-world credit card transactions, capturing authentic patterns, noise, and behavior that are representative of actual fraud attempts. One key advantage of this dataset is its benchmark-friendly nature, making it suitable for model development, evaluation, and comparison across different algorithms. However, the dataset is limited in sizes and most of its features have been transformed using Principal Component Analysis (PCA), which can obscure the original meaning of the variables and complicate interpretability.

To overcome the limitations of the empirical dataset, we incorporate a synthetic dataset. Unlike real-world data, synthetic data can be generated at virtually unlimited scale, making it essential for evaluating the performance and scalability of a fraud detection system built to handle big data. This large volume is crucial for simulating real-world operational loads and ensuring that the system remains responsive and accurate under high-throughput conditions. Moreover, synthetic datasets offer fine-grained control over transaction attributes, support balanced class distributions, and allow for the simulation of diverse fraud scenarios. While they may not capture the full complexity of real user behavior, their scalability and flexibility make them invaluable for testing system robustness, optimizing resource usage, and visualizing model behavior at scale.

Using both datasets allows us to leverage their complementary strengths. Overall, combining empirical and synthetic data results in a more comprehensive and effective fraud detection system than relying on either dataset alone.

### *5.2.1. Empirical Dataset*

Our project incorporates the "Credit Card Fraud Detection" dataset released by the Machine Learning Group at Université Libre de Bruxelles (ULB), available on Kaggle[2]. This empirical dataset contains real-world anonymized credit card transactions made by European cardholders in September 2013. It includes a total of 284,807 transactions, of which only 492 are labeled as fraudulent, representing a severe class imbalance of approximately 0.17%.

Each transaction is described by 30 features. Among them, 28 are anonymized principal components (V1 to V28) obtained via PCA transformation, along with the `Time`, `Amount`, and `Class` fields. The `Class` variable serves as the binary label, where 1 denotes a fraudulent transaction and 0 a legitimate one. The use of PCA ensures that sensitive information is obscured, maintaining confidentiality while preserving useful patterns for machine learning.

This dataset was selected due to its authenticity and relevance to real-world fraud detection challenges. Despite its smaller size and anonymized fields, it offers invaluable insights into genuine fraud behaviors. Its extreme imbalance also provides a realistic test bed for evaluating model performance under production-like conditions, where frauds are rare but costly.

By combining this empirical dataset with a large-scale synthetic counterpart, we ensure both realism and scalability in our fraud detection framework. This allows us to train, test, and refine models in a way that balances fidelity to real-world conditions with the flexibility of synthetic data augmentation.

### *5.2.2. Synthetic Dataset*

In addition to empirical data, our project leverages the "Fraud Detection" dataset by `Kartik2112`, available on Kaggle[3]. This dataset simulates real-world financial transactions and includes over 6 million records, each labeled as either fraudulent or legitimate. It provides a rich set of features such as transaction amount, merchant category, origin and destination accounts, customer demographics, geolocation data, and timestamps.

The dataset spans from January 1, 2019 to December 31, 2020 and is synthetically generated using the `faker` library and the Sparkov data generation tool. While the fields are designed to mimic real-world financial transactions, the data does not represent any actual individuals or institutions. This ensures the dataset is ethically sound and suitable for academic research and experimentation.

This dataset was chosen for its strong alignment with the requirements of fraud detection in a Big Data context. It is large-scale, well-structured, and information-rich, making it ideal for processing with tools like Hadoop and Spark. Furthermore, its synthetic nature supports safe expansion and scenario testing without privacy concerns.

## *5.3. Data Schemas*

### *5.3.1. Empirical Dataset*

This dataset consists of 30 attributes describing the characteristics of each transaction. Most of the fields have been transformed to protect sensitive information. Specifically, 28 of the features are the result of a dimensionality reduction process known as Principal Component Analysis (PCA), which helps preserve underlying patterns while anonymizing the original variables. These are labeled from `V1` to `V28`.

In addition to these, there are two straightforward fields: `Time`, which represents the number of seconds elapsed since the first recorded transaction in the dataset, and `Amount`, which reflects the monetary value of each transaction. The final column, `Class`, serves as the label, where a value of 1 indicates a fraudulent transaction and 0 indicates a legitimate one.

Each entry in the dataset corresponds to one individual transaction. Even though the dataset does not contain customer identifiers, merchant information, or location data, it still allows for meaningful analytical work. The PCA-transformed features capture behavioral and transactional anomalies, which are essential for training fraud detection models.

### *5.3.2. Synthetic Dataset*

The dataset consists of 23 columns, which capture various aspects of each transaction. The features can be grouped as follows:

- **Transaction Info:** `trans_date_trans_time`, `amt`, `category`, `merchant`, `trans_num`, `unix_time`.

[2]`https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud`

[3]`https://www.kaggle.com/datasets/kartik2112/fraud-detection`

- **Customer Info:** `cc_num`, `first`, `last`, `gender`, `dob`, `job`, `street`, `city`, `state`, `zip`, `lat`, `long`, `city_pop`.
- **Merchant Info:** `merch_lat`, `merch_long`.
- **Target Variable:** `is_fraud` (binary label: 1 = fraudulent, 0 = legitimate).

Each row in the dataset represents an individual transaction. The structure supports advanced feature engineering, such as:

- **Temporal analysis:** e.g., extracting hour, day of week.
- **Geospatial analysis:** e.g., distance between customer and merchant.
- **Demographic insights:** e.g., age, gender, population density.

This well-structured format enables robust modeling and facilitates both batch and real-time machine learning pipelines in the context of fraud detection.

### *5.4. Data Generation for Synthetic Dataset*

From the chosen Kaggle dataset, which contains simulated credit card transactions for 1,000 customers over January 1, 2019 to December 31, 2020, our team is able to generate additional data to scale up for Big Data processing.

The dataset was created using the Sparkov Data Generation tool, which defines transaction behaviors based on customer profiles, using the faker library to simulate realistic transaction details. Transactions are created according to the related profile, such as adults_2550_female_rural.json, which defines simulation characteristics for adult females aged 25-50 in rural areas. Each profile defines the parameter for generating simulated transactions, outlines the behavior and characteristics of customers, including their frequency, distribution across time, spending categories, amounts, and other preferences. Some key sections are:

- Range of Average Transactions per Day: Defined by a minimum (min) and maximum (max) number of transactions, specifying the typical daily transaction range for the profile.
- Weighting for Transaction Likelihood: Transaction probability varies by:
  - Day of the Week: Higher weights for Saturday and Sunday reflect increased transaction likelihood on weekends.
  - Time of Year: Spending spikes during holidays, dips post-holidays, and rises moderately in summer, aligning with seasonal consumer trends.
  - Specific Years: Weights indicate consistency or variation in transaction behavior across defined years.
- Category Prioritization: Essential categories like gas/transpor and groceries have higher weights, indicating frequent spending, while travel has a lower weight, suggesting it is less common for this profile (e.g., rural females aged 25–50).
- Transaction Amount Patterns: Essential categories like gas and groceries have predictable amounts with low variability, while discretionary categories like travel and online shopping allow for outliers, such as expensive trips or large online orders.

We use the same simulation logic with 1,000 customers over the period from January 1, 2019 to December 31, 2020, following the structure of the original dataset. This approach enables us to generate additional transactions that align with the same statistical distributions, ensuring consistency in both schema and behavior. As a result, we can efficiently create a much larger dataset—potentially several gigabytes in size—within minutes, making it well-suited for Hadoop and Spark workflows.

## 6. Fraud detection model

### *6.1. Related works*

Fraud detection in financial transactions has attracted significant research attention, with numerous models and preprocessing techniques being proposed. In [8], the authors experimented with a range of classical machine learning algorithms such as Decision Tree, K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machine (SVM), Random Forest, and XGBoost. These models achieved impressive accuracy rates above 99.9%, while F1-scores ranged between 75% and 85%. Additionally, Convolutional Neural Networks (CNNs) were evaluated on both imbalanced and balanced datasets, showing performance varying from 0.9 to 0.99 depending on architecture depth and number of training epochs. The study concluded that deep learning models performed better with larger datasets compared to traditional methods. In [9], the impact of dimensionality reduction techniques, including Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Autoencoders, was studied in conjunction with the SMOTE oversampling technique. The authors defined four experimental branches, combining different reduction methods with or without SMOTE, to understand how these preprocessing steps influence fraud detection accuracy. The study in [10] proposed using DevNet for anomaly detection across multiple datasets. It demonstrated superior performance compared to baseline methods. Preprocessing steps included imputing missing values with mean values and applying one-hot encoding to categorical features. In [11], the authors explored various deep learning and ensemble methods, notably AutoEncoder, CNN, LSTM, and LightGBM. These were combined with SMOTE to manage class imbalance, yielding effective results in credit card fraud detection. A novel approach using a Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) with an attention mechanism was proposed in [12]. The model aimed to capture long-term dependencies in transaction sequences while emphasizing important time steps through attention, improving detection accuracy. Domain adaptation was addressed in [13], where the authors designed a model to leverage knowledge from related datasets. SMOTE and GAN-based oversampling were used to address class imbalance. They compared SVM, CNN, and their proposed domain-adaptive model, finding that while SVM achieved the highest AUC, it was significantly more computationally expensive. In [14], several baseline classifiers such as Decision Tree, Random Forest, and Extra Trees were enhanced using AdaBoost. This combination led to improved classification performance, showcasing the effectiveness of ensemble learning techniques. Feature engineering was the focus in [15], which used Genetic Algorithms for feature selection. The method generated five optimal feature vectors, which were then evaluated across various machine learning models to verify their effectiveness in improving detection. Finally, [16] assessed basic models such as Logistic Regression, Neural Networks, and SVM, but did not delve into advanced preprocessing or data balancing techniques, limiting the study's contribution in terms of methodology.

Overall, the literature shows a strong trend toward integrating data preprocessing techniques (e.g., SMOTE, dimensionality reduction), advanced models (e.g., CNN, LSTM, attention mechanisms), and domain adaptation strategies to enhance fraud detection performance.

## *6.2. Our approach*

### *6.2.1. Handling Class Imbalance*

To effectively address the inherent imbalance present in financial transaction datasets—where legitimate transactions vastly outnumber fraudulent ones—we planned to implement a variety of data balancing techniques as outlined in Table 4. These techniques aim to improve the model's ability to detect rare fraudulent activities without being overwhelmed by the majority class of legitimate transactions. For example, using SMOTE (Synthetic Minority Over-sampling Technique) allows for generating synthetic samples of the minority class (fraudulent transactions), helping to create a more balanced training set. Similarly, methods such as Random Undersampling reduce the number of majority class samples, thereby balancing the dataset and preventing the model from being biased toward the majority class. Adjusting class weights in the learning algorithms can also help by penalizing misclassification of fraudulent instances more heavily, encouraging the model to pay closer attention to fraud cases. Combining these approaches provides a comprehensive strategy to mitigate class imbalance, ultimately leading to more accurate and robust fraud detection performance.

Table 4: Techniques for Handling Imbalanced Fraud Detection Data

| **Technique** | **Description** |
|---|---|
| SMOTE | Generates synthetic samples of the minority class (fraud) to help the model detect fraud. |
| Random Undersampling | Reduces the majority class (non-fraud) to balance the dataset. |
| Class Weighting | Adjusts class weights to penalize misclassifications of fraudulent transactions more heavily. |

### *6.2.2. Model Selection*

To automate and streamline the process of model selection, we utilize the H2O AutoML framework. H2O AutoML is an open-source machine learning platform that automatically trains and tunes a large selection of models, including generalized linear models (GLM), gradient boosting machines (GBM), random forests, deep learning models, and stacked ensembles. It efficiently explores the model space using techniques such as cross-validation and hyperparameter tuning, and it ranks models based on performance metrics like AUC, logloss, or RMSE. This enables rapid identification of high-performing models without the need for extensive manual experimentation.

### *6.2.3. Evaluation Metrics*

Fraud detection is inherently challenging due to the imbalance between fraudulent and non-fraudulent transactions. Evaluating models based on accuracy alone is misleading, so we use the following metrics to assess performance effectively (Table 5):

Fraud detection is a binary classification task characterized by extreme class imbalance, where fraudulent transactions are rare but significantly more consequential than legitimate ones. Standard accuracy metrics fail to provide meaningful insights in such scenarios, as a naive classifier that always predicts the majority class (non-fraud) would achieve misleadingly high accuracy. Consequently, specialized performance metrics are required to rigorously evaluate model effectiveness. The four most critical metrics are precision, recall, F1-score, and ROC AUC, each of which provide distinct perspectives on model performance, as formalized below.

*Precision.* quantifies the model's ability to minimize false positives (Type I errors), defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

Table 5: Evaluation Metrics for Fraud Detection

| Metric | Description |
|---|---|
| Accuracy | Proportion of predicted cases that match actually fraudulent labels. In case of imbalance data, this metric is not very meaningful. |
| Precision | Proportion of predicted fraud cases that are actually fraudulent. Important when false positives are costly. |
| Recall | Proportion of actual fraudulent transactions identified by the model. High recall is critical to avoid missing fraud. |
| F1 Score | Harmonic mean of precision and recall. Balances the trade-off between false positives and false negatives. |
| ROC AUC | Measures the trade-off between the true positive rate and false positive rate. Useful for both classes. |

where $TP$ and $FP$ denote true positives and false positives, respectively. In fraud detection, high precision ensures that transactions flagged as fraudulent are indeed fraudulent, thereby reducing unnecessary customer interventions (e.g., declined payments or account freezes). This is particularly critical in domains where false alarms incur reputational damage or operational costs. However, optimizing for precision alone risks allowing undetected fraud (false negatives) to proliferate.

*Recall (Sensitivity).* measures the model's capacity to identify true fraud cases, expressed as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

with $FN$ representing false negatives. Maximizing recall is often a primary objective in fraud detection, as failing to detect fraudulent transactions (high $FN$) directly translates to financial losses. Regulatory frameworks, such as those in banking and insurance, may mandate minimum recall thresholds to ensure compliance. Nevertheless, recall-centric optimization can lead to excessive false positives if not balanced with precision.

*F1-Score.* harmonizes precision and recall via their harmonic mean:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

This metric is especially salient when the cost of false positives and false negatives must be jointly minimized. For fraud detection systems operating under resource constraints (e.g., limited manual review capacity), the F1-score provides a balanced criterion for threshold selection.

*ROC AUC.* evaluates the model's discriminative power across all possible classification thresholds by computing the area under the receiver operating characteristic curve. The ROC curve plots the true positive rate (recall) against the false positive rate ($FPR = \frac{FP}{FP+TN}$) at varying thresholds. A model with perfect discrimination achieves an AUC of 1, while random guessing yields an AUC of 0.5. In fraud detection, ROC AUC is invaluable for comparing models independently of threshold choices, as it summarizes performance across all operational conditions. High AUC indicates robust separation between the distributions of fraud and non-fraud transactions—a property critical for risk scoring systems.

*Metric Selection Trade-offs.* The choice of optimization metric depends on business objectives and operational constraints:

- *Recall-driven* strategies are prioritized when undetected fraud is prohibitively costly (e.g., high-value transactions).
- *Precision-driven* approaches prevail when false positives disrupt customer experience or incur review costs.
- *F1-score* is favored when a balance is mandated by resource limitations.
- *ROC AUC* guides model selection during development but requires threshold calibration for deployment.

Ultimately, these metrics are not mutually exclusive; a comprehensive evaluation should incorporate all four, with domain-specific weighting applied to align with organizational risk tolerance. In Table 6, we summarize the performances of previously studied works.

## 7. Monitoring and Performance

### 7.1. Monitoring

In our project, Grafana was employed as the central monitoring and visualization tool to provide real-time insights into system performance and fraud detection dynamics. The dashboard, shown in Figure 2, consolidates various critical indicators, including message latency, throughput, fraud event distribution, and geographical transaction patterns. These metrics enable operators and analysts to observe the behavior of the stream processing systems under different workloads and identify anomalies promptly.

To assess the comparative performance of Apache Flink and Apache Spark in a real-time processing context, we configured the dashboard to monitor both engines concurrently. This parallel measurement approach allows for a direct comparison of key performance indicators such as processing latency and system throughput. By doing so, we gained empirical evidence highlighting the trade-offs

Table 6: Summary of Machine Learning methods used for Fraud Detection problem

| Author | Methods | Best Model | Accuracy | Precision | Recall | F1-score | AUC |
|---|---|---|---|---|---|---|---|
| Fawaz Khaled Alarfaj et al.[8] | Tests ML models (DT, KNN, LR, SVM, RF, XGBoost); CNNs (on imbalanced & balanced data) | CNN using epoch size 35 | **0.999** | 0.932 | 0.775 | x | 0.929 |
| Mohammed Rashad Baker et al.[9] | 4 test setups using PCA, LDA, Autoencoder for dimensionality reduction with/without SMOTE to handle imbalance; branches compare DR methods and impact of oversampling. | Branch-2 Voting ensemble DR: PCA, Oversampling: x | 1 | **0.973** | 0.735 | **0.837** | x |
| Guansong Pang et al.[10] | DevNet tested on multiple datasets, showing strong performance in anomaly detection; preprocessing included mean imputation for missing values and one-hot encoding for categorical features. | DevNet (Deviation networks) | x | x | x | x | 0.98 |
| Yong Fang et al.[11] | Uses key techniques in credit card fraud detection include deep learning (AutoEncoder, CNN, LSTM) and ensemble learning (LightGBM), with SMOTE used to handle data imbalance. | LightGBM | 0.992 | x | **0.99** | x | 0.991 |
| Jeonghyun Hwang et al.[13] | Proposes a domain-adaptation model for fraud detection, with SMOTE and GAN-based oversampling, testing SVM, CNN, and the proposed model. | SVM + Oversampling using GAN | x | x | x | x | **0.996** |
| Emmanuel Ileberi et al.[14] | Tests basic models like Decision Tree, Random Forest, and Extra Tree, each combined with AdaBoost to enhance classification performance and significantly improve accuracy. | Extra Tree-AdaBoost | 0.9998 | x | x | x | x |
| Emmanuel Ileberi et al.[15] | Uses Genetic Algorithm for feature selection, yielding 5 optimal feature vectors, and evaluates its effectiveness with basic models in various scenarios. | For v1 vector: RF | 0.9994 | 0.8969 | 0.7699 | 0.8285 | 0.96 |
| Shahnawaz Khan et al.[16] | Evaluates basic models including Logistic Regression, Neural Networks, and Support Vector Machine, without specifying other processing techniques. | SVM | 0.9994 | 0.8767 | 0.7805 | 0.8258 | 0.94 |

between high-throughput (Spark) and low-latency (Flink) architectures, supporting informed decision-making for deployment in latency-sensitive versus volume-intensive financial environments.

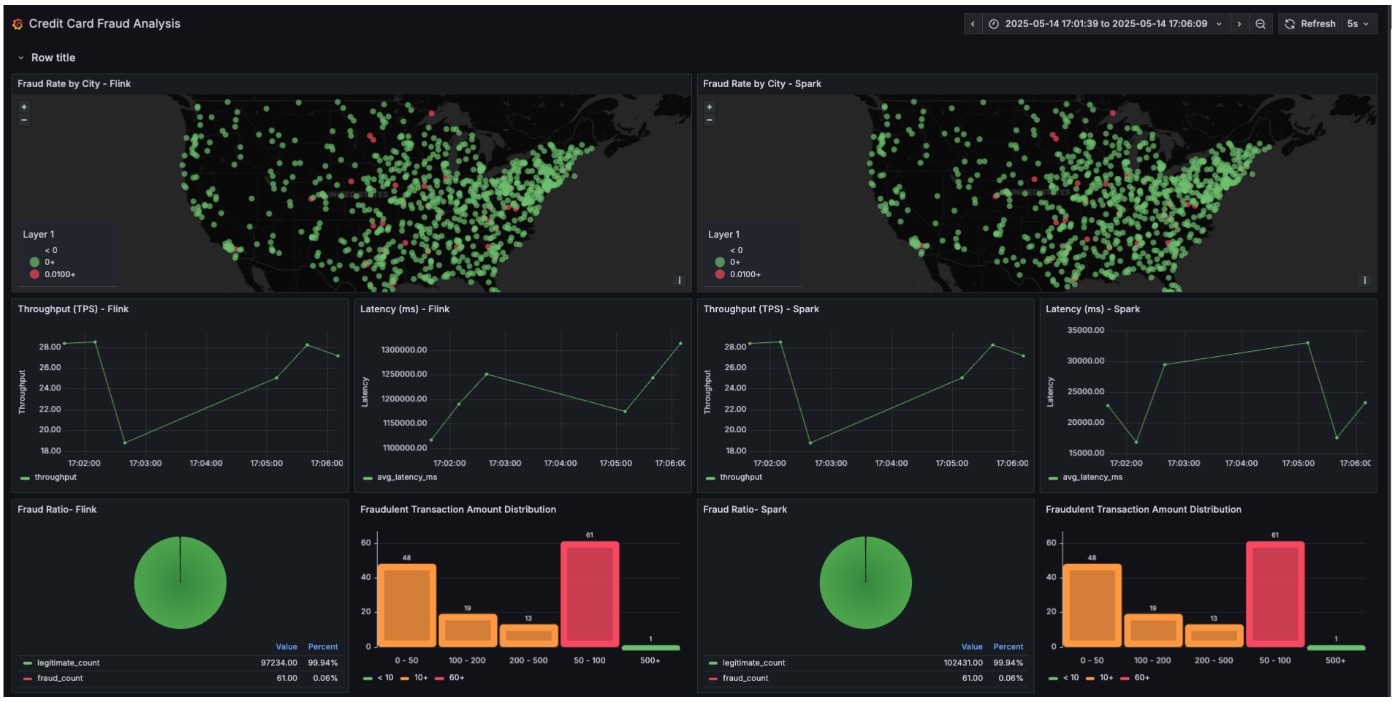

Figure 2: Monitoring Grafana Dashboard

### *7.2. Prediction Performance*

The **Stacked Ensemble GLM model** used in our system is generated using **H2O AutoML**. The stacking process is described as follows:

- Base Model Training: Individual base learners, including Gradient Boosting Machine (GBM) and XGBoost, are independently trained on the training dataset.
- Out-of-Fold Predictions: A cross-validation strategy is applied to generate out-of-fold predictions from the base models. These predictions are used as input features for the next stage.
- Meta-Learner Training: A Generalized Linear Model (GLM) acts as the meta-learner. It takes the out-of-fold predictions as inputs and learns how to optimally combine the outputs from GBM and XGBoost to produce the final prediction.

This ensemble strategy effectively leverages the strengths of individual base learners while mitigating their respective weaknesses, resulting in improved overall predictive performance. As presented in Table 7, the H2O Stacking Model demonstrates competitive AUC scores when compared with state-of-the-art methods. Although the precision and recall values are relatively lower, this trade-off is justifiable given that the ensemble is built upon statistical machine learning models, which are significantly less complex than deep learning architectures such as CNNs. Consequently, the proposed approach strikes a practical balance between model simplicity and predictive capability.

### *7.3. Latency - Throughput*

#### *7.3.1. Methodology*

- **Latency Measurement**:
  - Each Kafka producer tags incoming transactions with an `ingestionTime` timestamp at the moment of ingestion.
  - As the transaction flows through the pipeline and reaches the alert engine, an `alertTime` timestamp is recorded at the moment the alert is generated.
  - The end-to-end processing latency is then calculated as:

$$\text{Latency} = \texttt{alertTime} - \texttt{ingestionTime}$$

- **Throughput Measurement**:
  - We measure the number of completed transactions within fixed 30-minute intervals.
  - The throughput metric is defined as:

$$\text{Throughput} = \frac{\text{Number of Completed Transactions}}{\text{Window Duration (30 minutes)}}$$

#### *7.3.2. Results*

The results presented in Table 8 illustrate the varying performance characteristics of Spark and Flink under different transaction loads. As the number of transactions

Table 7: Model Evaluation Metrics

| **Model** | **AUC** | **Accuracy** | **Precision** | **Recall** | **F1-score** |
|---|---|---|---|---|---|
| CNN Model | 0.929 | 0.999 | 0.932 | 0.775 | x |
| LightGBM [9] | 0.991 | 0.992 | x | x | x |
| **H2O stacking model** | **0.996** | **0.9959** | **0.5995** | **0.5953** | **0.5974** |

generated by producers increases or decreases, the system exhibits a range of throughput and latency levels.

Fundamentally, the system must trade off between throughput and processing latency. Spark, for example, achieves high throughput - up to 111 TPS, but this comes at the cost of increased latency, which can reach as high as 40 seconds. This increase is largely due to Spark's micro-batching architecture, which makes it less suitable for scenarios where ultra-low latency is required.

When the transaction load is relatively low, a switch to Flink can offer significant latency improvements. Although Flink's maximum throughput in this setup is only 8 TPS, it delivers much lower latency, which is 6 seconds at peak, and as low as 5 seconds at 6 TPS. Notably, at 6 TPS, Flink's latency is approximately one-third of Spark's at the same throughput level.

These findings suggest that while Spark is advantageous for high-throughput workloads, Flink becomes the better choice when latency sensitivity is more critical than throughput.

| **Streaming Tech** | **Throughput (TPS)** | **Latency (s)** |
|---|---|---|
| Spark | 111 (Max) | 40 |
| Spark | 28.5 | 25 |
| Spark | 8 | 20 |
| Spark | 6 | 15 |
| Flink | 8 (Max) | 6 |
| Flink | 6 | 5 |

Table 8: System's performance results

### *7.4. Scalability*

#### *7.4.1. Methodology*

To validate the scalability of the system, we vary the number of Kafka brokers, Flink workers and Spark workers independently while keeping the load generation logic and evaluatingng performance metrics using the previously described methodology.

#### *7.4.2. Results*

Overall, scaling the number of instances for each module (Kafka, Flink, and Spark) does not disrupt the system's operational workflow. However, such changes do affect system performance, particularly in terms of latency and throughput, most notably during transitional periods.

## 8. Conclusion

In conclusion, this project successfully designed and implemented a real-time data processing architecture capable of handling financial transaction data at scale. By integrating both Apache Spark and Apache Flink, the system leveraged the strengths of each technology: Spark for high-throughput workloads and Flink for low-latency processing, thereby allowing flexible adaptation to different performance requirements.

An anomaly detection pipeline, incorporating machine learning models, was developed to identify potential fraudulent activities with accuracy and efficiency. The architecture was tested for scalability, showing the ability to sustain high transaction rates while balancing throughput and latency trade-offs.

To support monitoring and analysis, an interactive dashboard was also developed, providing real-time visibility into key system metrics and fraud detection outcomes. Overall, the system demonstrates a robust, scalable, and efficient approach to processing and analyzing financial data in real time.

## Members' main contributions

- **Nguyen Thanh Dao (Team Leader)**: Led the overall system design and workflow, coordinated team collaboration, and contributed to report writing.
- **Nguyen Van Thien**: Configured Data producers, the Kafka cluster, connectors, and Flink.
- **Nguyen Huu Phuong**: Configured Spark and conducted model training using H2O Sparkling Water.
- **Pham Ngoc Linh**: Conducted research on Kafka and relevant datasets; contributed to report writing.
- **Tran Thu Thuy**: Designed and implemented Grafana dashboards, researched datasets, and conducted literature reviews on machine learning models.

## References

[1] Fraud.net, Fraud detection using machine learning vs. rules-based systems (2023).
URL `https://www.fraud.net/resources/fraud-detection-using-machine-learning-vs-rules-based-systems`

[2] IJERD, Real-time fraud detection in high-speed financial environments, International Journal of Engineering Research and Development (2023).
URL `https://ijerd.com/paper/vol20-issue11/201111781187.pdf`

[3] S. B. Journal, The evolution of financial fraud and the need for adaptive detection (2023).
URL `https://thesciencebrigade.com/JAIR/article/view/402`

[4] Redpanda.com, Kafka Connectors - Overview, use cases, and best practices (2014).
URL https://www.redpanda.com/guides/kafka-cloud-kafka-connectors

[5] Confluent, Google Cloud BigQuery Sink Connector V2 for Confluent Cloud Quick Start.
URL https://docs.confluent.io/cloud/current/connectors/cc-gcp-bigquery-storage-sink.html

[6] M. L. G. ULB, Credit Card Fraud Detection (2021).
URL https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

[7] K. Shenoy, Credit Card Transactions Fraud Detection Dataset (2020).
URL https://www.kaggle.com/datasets/kartik2112/fraud-detection

[8] 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.

[9] Ensemble Learning with Supervised Machine Learning Models to Predict Credit Card Fraud Transactions — IIETA — iieta.org, https://www.iieta.org/journals/ria/paper/10.18280/ria.360401, [Accessed 13-05-2025].

[10] G. Pang, C. Shen, A. van den Hengel, Deep anomaly detection with deviation networks, CoRR abs/1911.08623 (2019). arXiv:1911.08623.
URL http://arxiv.org/abs/1911.08623

[11] C. H. Yong Fang, Yunyun Zhang, Credit card fraud detection based on machine learning, Computers, Materials & Continua 61 (1) (2019) 185–195.
URL http://www.techscience.com/cmc/v61n1/23107

[12] J. Femila Roseline, G. Naidu, V. Samuthira Pandi, S. Alamelu alias Rajasree, D. Mageswari, Autonomous credit card fraud detection using machine learning approach, Computers and Electrical Engineering 102 (2022) 108132. doi:https://doi.org/10.1016/j.compeleceng.2022.108132.
URL https://www.sciencedirect.com/science/article/pii/S0045790622003822

[13] J. Hwang, K. Kim, An efficient domain-adaptation method using gan for fraud detection, International Journal of Advanced Computer Science and Applications 11 (11) (2020). doi:10.14569/IJACSA.2020.0111113.
URL http://dx.doi.org/10.14569/IJACSA.2020.0111113

[14] E. Ileberi, Y. Sun, Z. Wang, Performance evaluation of machine learning methods for credit card fraud detection using smote and adaboost, IEEE Access 9 (2021) 165286–165294. doi:10.1109/ACCESS.2021.3134330.

[15] A machine learning based credit card fraud detection using the GA algorithm for feature selection - Journal of Big Data — journalofbigdata.springeropen.com, https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00573-8#citeas, [Accessed 13-05-2025].

[16] S. Khan, A. Alourani, B. Mishra, A. Ali, M. Kamal, Developing a credit card fraud detection model using machine learning approaches, International Journal of Advanced Computer Science and Applications 13 (3) (2022). doi:10.14569/IJACSA.2022.0130350.
URL http://dx.doi.org/10.14569/IJACSA.2022.0130350