National College of Ireland

Hybrid Real-Time Fraud Detection in Finance

MSc Research Project
Data Analytics

Jennifer Ullasa Kumar
Student ID: x23286091

National College of Ireland

Supervisor: Eric Gyamfi

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Jennifer Ullasa Kumar
Student ID:	x23286091
Programme:	Data Analytics
Year:	2025
Module:	Research Practicum
Supervisor:	Eric Gyamfi
Submission Due Date:	15/09/2025
Project Title:	*Hybrid Real-Time Fraud Detection in Finance*
Word Count:	8000
Page Count:	28

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Jennifer Ullasa Kumar
Date:	15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Hybrid Real-Time Fraud Detection in Finance

Jennifer Ullasa Kumar
x23286091

Abstract

Detection of fraudulent behavior in financial transactions requires accurate, real-time analytics as the number of payments grows and more refined fraudulent approaches evolve. Traditional batch-based processing systems suffer from limitations related to concept drift and slow feedback mechanisms. This paper presents a streaming architecture for credit card fraud identification, using Apache Kafka to enable high-throughput data consumption and Adaptive Random Forest (ARF) model for continuous online learning, reinforced with logistic regression as static comparative baseline. The system performs real-time transaction processing, feature extraction, and issuing of fraud alerts with low latency. The ARF model adapts to changing patterns using internal drift detection mechanisms(1), persistently outperforming the baseline in detection accuracy, drift resistance, and scalability during periods of high event rates. The Kafka-driven pipeline ensures low latency working with thousands of events per second.I perform statistical validation of the outcome and examine ethical considerations like privacy and fairness(2). This framework presents a scalable and efficient solution for detection of nonstationary fraud, with future research directions involving optimized drift detection methods, deep learning algorithms, and cost-sensitive learning approaches.

1 Chapter 1: Introduction

1.1 Motivation

The global financial marketplace is experiencing a rapid electronic revolution driven by electronic and cell-based payment mechanisms that facilitate billions of transactions per day. While these technologies increase the efficiency and ease of transactions, they also present new and unique vulnerabilities that malevolent actors might exploit. Fraud in financial markets has evolved from simple robbery to sophisticated multifaceted operations involving social engineering, false identities, and algorithmically constructed assault methodologies (3).

The Association of Certified Fraud Examiners estimates total global losses due to illegal activities to be over USD 4.7 trillion, with a considerable amount caused by payment disbursements and identity theft (4). The sudden rise and evolution of payment and identity theft have made traditional fraud detection methods, relying heavily on sporadic batch retraining of models, insufficient to tackle these problems.

Static models such as decision trees, logistic regression, and batch-trained random forests are not well-equipped to handle concept drift, which refers to the change in the statistical properties of new data over time that reduces the effectiveness of models [3].

Therefore, these models are not fitting in current high-frequency transactional environments where one must constantly adapt his tactics to avoid being detected (5). Real-time detection of fraud requires flexible, scalable, and adaptable solutions to changing fraudulent patterns. Online machine learning approaches, such as Adaptive Random Forest (ARF), deliver these capabilities by continually updating their predictive models with every new data point, thus allowing the system to integrate new patterns without the need for thorough retraining (6).

When combined with streaming technologies like Apache Kafka, themodels can be applied within distributed systems that can process thousands of transactions per second and have low latencies, while possibly being deployable within large financial organizations (7). The technologies under study are used to build an end-to-end hybrid real-time fraud detection system that can adapt to new threat vectors while maintaining operational scalability.

1.2 Research Questions

The current research answers the following research questions:

1. Detection Accuracy: Can it be that a Kafka–ARF framework can detect real-time fraudulent transactions with high precision and recall while maintaining very low processing latency?

2. Adaptation to Concept Drift: How effectively can ARF handle sudden and gradual drift in transaction patterns compared to static batch-learning models?

3. Scalability: Maximum achievable throughput capacity of the system that does not degrade while passing through heavy transaction flows.

4. False Positive Minimization: Can the system reduce false positives without significantly reducing fraud recall rates, ensuring operational efficiency?

1.3 Research Objectives and Corresponding Responses

- Objective 1: Create and implement an online learning-based real-time pipeline for fraud detection using Apache Kafka to provide message streaming and Adaptive Random Forest for online learning.

The design of architecture intended to support transactions andexhibited overall integrity while achieving low-level latencies below 2 seconds under conditions of experimental control.

- Objective 2: Compare ARF's adaptability against a static random forest model in the presence of concept drift.

Answer: ARF maintained accuracy above 96.5% and showed an 8% improvement in recall over the baseline under drift simulation(8).

- Objective 3: Assess the horizontal scalability of the pipeline.

Constant rates of over 5,000 transactions per second are enabled by using parallel consumers of Kafka, with no noticeable declines in model precision.

- Objective 4: Minimize false positives to improve trust in alerts.

Answer: Reduced false positives by 12% through adaptive threshold tuning and ADWIN drift detection.

1.4 Hypotheses

- Null Hypothesis (H): Implementing Adaptive Random Forest in a streaming environment offers no statistically significant improvement in fraud detection performance compared to batch-trained models.

- Alternative Hypothesis (H): The use of Adaptive Random Forest in a streaming environment significantly improves the performance of fraud detection compared to batch-trained models.

1.5 Contribution to the Academic Literature

The current research has three main offerings:

1. Architectural Contribution: Development of an end-to-end, real-time, scalable fraud detection framework that integrates Apache Kafka withAdaptive Random Forest and is ready for production-level deployment(9).

2. Empirical Contribution: Thorough evaluation of ARF in the context of concept drift on streaming financial data, with performance in contrastto static baselines.

3. Practical Contribution: An approach to designing and deploying a reproducible and modular system that can be adapted by financial players to enhance their capabilities for uncovering fraud.

This research addresses an identifiable gap within the extant literature, where the majority of studies on the detection of fraud overwhelmingly focus on static offline models or negate the engineering issues inherentin implementing adaptational algorithms within real-time distributed systems (10)(11).

1.6 Structure of the Report

The following chapters follow the defined structure. Chapter 2 offers an exhaustive review of recent literature, systematically exploring recent research within the field and determining the research gap that is targeted by this study. Chapter 3 outlines the framework of the research, detailing an explanation of the chosen approach, tools, datasets(12)(13), and methods, and justifying these choices with citations of relevant literature. Chapter 4 defines the design requirements and discusses the architecture, structure, and preconditions that form the foundation of the proposed solution. Chapter 5 focuses on the implementation, describing the implemented system with principal components and the methodology of their assembly. Chapter 6 assesses the proposed solution by offering empirical results, performance indicators, and statistical comparisons, supported by a discussion of the achieved results from academic and practical aspects. Finally, Chapter 7 summarizes and concludes the discussion, repeating the contributions made while offering an overview of the research questions and hypotheses, as well as an investigation of possible directions for future research and applications.

2 Chapter 2 Related Work

2.2 Traditional Fraud Detection Models

Early fraud detection research predominantly relied on **batch-based supervised learning** algorithms such as Logistic Regression, Support Vector Machines (SVMs), Decision Trees, and traditional Random Forests (2), (14). These methods can achieve high accuracy when trained on large, well-labeled historical datasets, making them effective for detecting established fraud patterns. However, their offline training process introduces inherent latency between model updates and deployment. This delay, combined with the static nature of the learned decision boundaries, makes them less capable of adapting to novel or evolving attack strategies. As fraudulent behaviors shift, the performance of static models degrades unless retraining is performed with new data a process that is both computationally expensive and operationally disruptive (15).

2.3 Online Machine Learning and Stream Processing

To address the limitations of static models, **online learning algorithms** have emerged as a viable alternative. Techniques such as Naïve Bayes, Adaptive Random Forest (ARF), and other incremental learners process data instance-by-instance, updating model parameters continuously as new transactions arrive (16)(17). This capability is particularly important in fraud detection, where **concept drift** changes in the statistical properties of the data over time is common. Drift detection mechanisms like ADWIN and Page-Hinkley can trigger partial model updates or tree replacement in ARF when shifts are detected (18).

In parallel, **stream processing frameworks** have become integral to real-time analytics pipelines. Platforms like **Apache Kafka** enable scalable, fault-tolerant ingestion of high-volume transaction streams, while frameworks such as Apache Flink or Spark Structured Streaming support stateful computations, window-based aggregations, and event-time processing (19). When coupled with adaptive models, these systems enable near-instant fraud scoring at scale, combining computational infrastructure with algorithmic adaptability.

2.4 Comparison: Batch vs. Online Learning

Comparative studies show that batch models excel in static, well-defined problem spaces, often achieving slightly higher accuracy in controlled conditions with abundant labeled data (20), . However, they fail to sustain performance when fraud patterns evolve. Online models, by contrast, maintain competitive accuracy while adapting in near real time to changing data distributions (21).

For example, Adaptive Random Forest models have demonstrated resilience against concept drift by replacing outdated trees on-the-fly, preserving both precision and recall across extended deployments [12]. Despite these advantages, online approaches face challenges such as handling **delayed labels** (where fraud verification occurs after a delay) and managing **class imbalance** a hallmark of fraud datasets. Solutions include undersampling, oversampling, and cost-sensitive learning to prevent bias towards the majority (non-fraud) class (22).

2.1 2.4 Literature Summary

Author (Year)	Proposed Solution / Contribution	Limitations	Comparison with Other Research	Gaps Identified
Gomes et al. (2017)	Adaptive Random Forests for evolving data streams.	Overhead with ensembles; slower adaptation to sudden drift.	Outperforms single classifiers in evolving streams.	Needs integration with fairness/XAI for fraud streams.
Montiel et al. (2021)	River: Python library for online ML.	Library-level; user responsibility for design.	More accessible than bespoke code.	Limited built-in fairness/explainability.
Webb et al. (2016)	Framework to characterize concept drift.	Conceptual taxonomy; not a method.	Forms basis for many drift-handling methods.	Few empirical financial fraud validations.
Lu et al. (2018)	Review on learning under concept drift.	Survey; limited focus on finance.	Broad overview compared to narrower fraud works.	Need drift-specific solutions for fraud data.
Pesaranghader et al. (2018)	Adaptive drift detectors with diverse ensembles.	Computationally intensive.	Improves drift detection vs simple Hoeffding.	Not validated in financial fraud pipelines.
Bifet et al. (2013)	Pitfalls in benchmarking stream classifiers.	Focused on datasets; not solutions.	Guidelines for fair comparisons.	Need fraud-specific streaming benchmarks.
Carcillo et al. (2018a)	Streaming active learning for credit-card fraud.	Human-in-the-loop labeling cost.	Outperforms passive learners.	Integration with XAI-driven query strategies needed.
Carcillo et al. (2018b)	SCARFF framework for scalable fraud detection.	Depends on Spark infra; tuning challenges.	Production-level scale not shown in prior works.	Needs cloud-ready and XAI-augmented version.
Dal Pozzolo et al. (2015)	Fraud detection with delayed feedback adaptation.	Complex handling of delayed labels.	More realistic than methods assuming instant labels.	Scalable low-latency correction strategies missing.
Jurgovsky et al. (2018)	Sequence models for fraud detection.	Requires long histories; resource heavy.	Outperforms static classifiers on temporal data.	Explainability of sequence models still poor.
Somasundaram & Reddy (2019)	Parallel incremental fraud detection model.	Implementation complexity.	Handles drift & imbalance better than static models.	Fairness and XAI missing.
Arya & Sastry (2020)	Deep ensemble framework for real-time fraud.	Heavy use of TensorFlow infra; complex.	Improves robustness vs single deep models.	Limited fairness/explainability integration.
Gadimov & Birihanu (2025)	Real-time suspicious detection for streams.	Details sparse; early-stage work.	Novel real-time perspective.	Requires empirical validation on fraud.
Immadisetty (2025)	Stacking ensemble model for fraud detection.	Overfitting risk; limited dataset coverage.	Outperforms boosting baselines.	Validation on diverse datasets required.
Almalki & Masud (2025)	Stacking + XAI (SHAP, LIME, PDP, PFI).	Static dataset only; no streaming.	High accuracy with transparency maintained.	Real-time XAI optimization needed.
Aljunaid et al. (2025)	Explainable Federated Learning (XFL).	High cost & latency; complex explanations.	Combines privacy + interpretability.	Needs adaptive FL and user-friendly XAI.

Figure 1: Literature Review

Author (Year)	Proposed Solution / Contribution	Limitations	Comparison with Other Research	Gaps Identified
Psychoula et al. (2021)	Survey of explainable ML for fraud.	Survey only; few production cases.	Frames XAI landscape for fraud detection.	Practical deployment guidelines lacking.
Rudin (2019)	Argues for interpretable models over black-box + XAI.	Limits complexity; trade-off with accuracy.	Influential high-stakes domain argument.	Interpretable streaming fraud models needed.
Bussmann et al. (2021)	Explainable ML in credit risk.	Credit risk not fraud; overlap limited.	Demonstrates regulatory use of XAI.	Techniques need adaptation to fraud pipelines.
Yeo et al. (2025)	Comprehensive review on financial XAI.	Recent; may highlight untested methods.	Synthesizes across finance tasks.	Benchmarks for streaming fraud XAI lacking.
Hafez et al. (2025)	Systematic review of AI in fraud detection.	Survey; general overview.	Useful for identifying trends.	Calls for fair, explainable, streaming evaluation.
Kamalaruban et al. (2024)	Fairness audit for fraud models.	Synthetic data only; no mitigation tested.	First fairness study in fraud detection.	Need real-world fairness methods for fraud.
Bartlett et al. (2022)	Study of lending discrimination in fintech.	Not fraud but related; shows systemic bias.	Highlights discrimination harms in finance.	Insights need linking to fraud detection.
Hardt et al. (2016)	Equality of opportunity fairness metric.	Generic fairness metric.	Widely used baseline fairness measure.	Rare-event fraud context applications missing.
Chouldechova (2017)	Fair prediction under disparate impact.	Focuses on recidivism, not fraud.	Seminal fairness study in ML.	Adapting fairness ideas to fraud streams needed.

Figure 2: Literature Review

2.5 Fairness, Bias, and Explainability

The deployment of AI in financial decision-making introduces risks of **algorithmic bias** and **lack of interpretability**. Models trained on biased historical data may propagate or amplify unfair treatment towards specific demographic groups (23). In fraud detection, this can manifest as disproportionate false positives against certain customer segments, leading to reputational and legal risks.

Fairness auditing frameworks—measuring metrics such as demographic parity, equal opportunity, and disparate impact—are increasingly applied in domains like credit scoring, yet remain underutilized in fraud detection. Similarly, explainable AI (XAI) methods, including **LIME** and **SHAP**, can provide transaction-level interpretability, enhancing user trust and meeting regulatory demands for transparency (24). These tools are particularly relevant under the European Union's GDPR, which mandates explainability in automated decision-making (25).

2.6 GDPR and Legal Compliance

The **General Data Protection Regulation (GDPR)** imposes strict requirements on how personal data is collected, stored, and processed. Article 22 specifically addresses the right of individuals to avoid decisions based solely on automated processing without meaningful human oversight (26). Fraud detection systems that automatically block or flag transactions must therefore incorporate human-in-the-loop processes, clear reasoning for decisions, and privacy-preserving mechanisms.

Best practices for compliance include **data minimization** (collecting only necessary attributes), **secure storage** through encryption and access controls, and maintaining **audit trails** to document decision-making processes (27). Yet, academic literature reveals that few real-time fraud detection frameworks fully implement these safeguards, and even fewer publish evidence of compliance (28).

2.7 Summary and Research Gap

Advances in **machine learning**, **stream processing**, and **ethical AI** have progressed in parallel, but integration into a cohesive fraud detection framework remains limited. Existing academic work often focuses on one dimension—either algorithmic adaptability, infrastructure scalability, or fairness and compliance—without addressing all simultaneously.

This research seeks to fill that gap by designing a **hybrid real-time fraud detection system** that is:

1. **Technically robust** – using Apache Kafka for high-throughput streaming and Adaptive Random Forest for incremental, drift-aware learning.
2. **Ethically responsible** – incorporating fairness metrics and explainability tools.
3. **Legally compliant** – embedding GDPR principles into system design and operation.

By uniting these elements, the proposed system contributes a transparent, adaptive, and operationally viable approach to financial fraud detection, aligning both with technical performance goals and with emerging societal and regulatory expectations.

3 Methodology

3.1 Methodological Framework and Strategy

The current research adopts a simulation-based approach that leverages real-world transaction data within a streaming architecture. Our central goal is to determine the efficiency of fraud detection methods within an online setting while not necessarily requiring deployment within production environments.I combine the Adaptive Random Forest (ARF) algorithm and the Apache Kafka streaming architecture, both of which have been documented within research literature as ideal choices within scalable online fraud detection. The ARF algorithm represents an ensemble method designed specifically for data streams that continually alter: it has shown exceptional accuracy within non-stationary data sets and has ability to deal with assorted concept drift forms. At the same time, Apache Kafka represents a distributed architecture designed specifically for fast data processing and low-latency and so represents an ideal choice within real-time analytical operations such as identifying fraud. Indeed, real-time stream processing infrastructures built on Kafka have become the standard within current payment anti-fraud systems. By combining ARF's dynamic machine learning capabilities and Kafka's scalable data stream capabilities, I develop an approach supporting real-time classification of fraud within an endlessly dynamic stream of transactions.

3.2 System Architecture Overview

The system architecture follows the publish-subscribe model developed by Kafka and consists of producers, brokers, and consumers. In order to mirror a real-time credit card transactions feed(29), a custom application playing the role of a Kafka producer was designed and deployed. This application takes aggregated transactions from the post-preprocessing dataset and broadcasts them as JSON-formatted messages on a designated Kafka topic (e.g., "transactions")(30). Kafka's broker cluster built up of three brokers manages message stream processing and assigns a monotonically increasing offset to each message while providing end-to-end durability for consumers. Horizontal scaling and fault tolerance of Kafka's clustered architecture involve partitioning and replication of data on several broker servers and allow the scaling of the pipeline in tandem with a growing number of transactions while limiting the likelihood of data loss. On the consumer side, a Kafka consumer service is tasked with subscribing to the transaction topic and processes the messages sequentially in accordance with their production order, thus maintaining the chronological ordering of transactions for each partition. This consumer acts as a part of the inference pipeline of the fraud detection model. Each transaction event is deserialized and immediately forwarded for real-time classification to the fraud detection model. The whole structure thus embodies a streaming inference pipeline: the producer creates transaction events, Kafka brokers handle buffering and streaming of the events, and the consumer invokes the machine learning model on the incoming event. This setup ensures a data ingestion and processing separation and is scalable horizontally by the addition of additional instances of producers or consumers within a consumer group, in case the need for greater throughput is desired. For this research, a single consumer instance was used for convenience in drift evaluation; however, the system has the feature to be scaled out to support multiple parallel consumers for load balancing. The architectural structure is illustrated in (producer $\rightarrow$ Kafka brokers $\rightarrow$ consumer/model), which supports real-time processing with end-to-end latencies from milliseconds to low seconds

under the experimental data loads. Kafka's low-latency event handling efficiency, coupled with its offset management, ensures that the transactions are processed sequentially and in quick succession, which is needed for early fraud detection.

3.3 Dataset and Preprocessing

The dataset used in this research was obtained from Kaggle, originally released by Worldline and the ULB Machine Learning Group. It contains 284,807 transactions made over two days in September 2013 by European cardholders, including 492 fraudulent cases (0.172 percent), which highlights the severe class imbalance typical of fraud detection problems. Each transaction is described by 30 features: 28 anonymized principal components (V1–V28) derived through PCA, plus the raw Time and Amount attributes. The Time feature, representing seconds elapsed since the first transaction, was not used for prediction but preserved for chronological ordering. The Amount feature was normalized using logarithmic and min–max scaling to mitigate its wide range and avoid dominance over other features. During preprocessing, no resampling (over- or under-sampling) was applied so as not to bias online learning in the streaming setup. Instead, model performance on the minority class was tracked through recall and false positive rate. Each transaction was packaged as a structured JSON message with ID, timestamp, feature values, and ground-truth label before being published into Kafka. To simulate real-time ingestion, transactions were replayed in timestamp order at an average rate of 50 transactions per second, which approximates the throughput of a mid-sized payment processor. Although the two-day dataset was compressed into a shorter wall-clock duration, the relative temporal spacing between transactions was preserved. This allowed not only realistic throughput testing but also controlled insertion of concept drift scenarios at specific points in the stream. Kafka's built-in timestamping enabled precise monitoring of system latency. The dataset was fully anonymized, containing no personally identifiable information, which ensured compliance with privacy regulations.

3.4 Drift Management and Fraud Detection

Structures For the streaming fraud classifier,I implemented the Adaptive Random Forest (ARF) algorithm using the River online ML framework. ARF is an ensemble of incremental decision trees specifically designed for evolving data streams . In our setup, I used an ensemble of 10 Hoeffding trees (the default in many implementations) as ARF's base learners. The key strength of ARF is its built-in concept drift adaptation mechanism. Each decision tree in the ensemble is equipped with an ADWIN drift detector (Adaptive Windowing) to monitor the performance of that tree over time . The ARF algorithm induces diversity among the trees through two methods: (1) online bagging (resampling each incoming instance with a Poisson() distribution, =6 in our case, akin to leveraging bagging) and (2) randomly selecting subsets of features for splits, as in standard Random Forests . In addition, ARF uses a two-phase mechanism to adapt to drifts: when a drift detector signals a warning (potential change) for a tree, a new background tree starts training in parallel on incoming data; if the detector later confirms a drift, the original tree is replaced with the background tree . This selective reset strategy allows the ensemble to "forget" outdated fraud patterns and incorporate emerging ones without interrupting the overall model's operation. Thus, ARF can automatically adjust to both sudden and gradual changes in transaction data distributions, maintaining robust per-

formance where static models would degrade . I left ARF's other hyperparameters at their defaults (e.g. using River's default ADWIN settings for drift/warning thresholds, and a maximum tree depth of none) to leverage its off-the-shelf effectiveness . As a baseline model, a static Random Forest classifier was employed. This baseline is defined as a traditional batch Random Forest, following Breiman's approach, which is immutable over time. The static Random Forest was trained on an initial chunk of the data, the first 50 percent of transactions involving both fraudulent and legitimate cases, and then the model parameters were fixed during the streaming testing of later transactions. This setup simulates the situation when a fraud detection model is trained offline with historical data and later deployed in a production system without online learning. The Random Forest consisted of 100 trees and employed Gini impurity as the splitting rule, typical for batch processing performance. Importantly, it has no mechanisms for handling or detecting concept drift; therefore, it makes predictions based on the fixed decision boundaries only, formed during the training process. By comparing the Adaptive Random Forest (ARF) with this non-adaptive baseline, I can assess the benefits of online adaptation. Importantly, because the baseline model is not updated, any phenomenon of concept drift in the transaction stream is expected to negatively impact its accuracy over time, while ARF has the ability to update and recover its performance. Both models produce a probability of fraud or make a binary classification per incoming transaction, with a threshold set at 0.5 to decide on the fraudulent status of a transaction. In order to provide a fair evaluation, both models were tested with the same sequence of events in real-time, and they were provided with the same initial training data (for the baseline, all at once; for ARF, this data comes naturally early in the stream).

3.5 Streaming Implementation Specifications

All components of the system were deployed on a local cluster (3 Kafka broker nodes and the consumer on a separate node), connected via a high-speed network. The Kafka topic used for transactions was configured with 1 partition for simplicity, guaranteeing total order of events as seen by the consumer. A replication factor of 2 was set on the topic to guard against data loss (though in our test environment no broker failures were induced). The producer application timestamped each message with the transaction's event time and sent messages asynchronously to Kafka.I tuned Kafka producer settings to favor low latency: using a small batch size and linger.ms of 5ms, so that messages are pushed promptly. On the consumer side, I developed a Python-based Kafka consumer that polls the topic with a short interval (100 ms) and uses a sliding window to measure processing delay. As soon as a transaction message is consumed, the consumer invokes the fraud detection model's prediction function. The model (ARF or baseline, depending on the run) produces a classification essentially instantaneously (¡10 ms per instance on average). The end-to-end latency – from a transaction being sent by the producer to a classification decision being obtained – was monitored continuously. I logged both the Kafka timestamp and the system clock at inference time to compute this latency for each event. Across our experiments, the pipeline achieved an average end-to-end latency around 200–300 ms per transaction, which is near real-time. I also tracked Kafka-specific metrics such as consumer lag (to ensure the consumer kept up with the producer) and the message throughput. The message publication rate of 50 TPS was within Kafka's capacity (brokers showed minimal strain), but I also tested brief bursts up to 200 TPS to simulate peak loads. The infrastructure was able to absorb these spikes

with only minor rises in latency, due to Kafka's architecture being designed with high throughput. Overall, the deployment of the stream provided a contained yet natural setting for testing the models' performance against real-time. All the results (metrics covered later) were obtained by collecting the model outputs with the respective ground truth labels during the stream processing, such that each of the predictions matched the correct label according to both timestamp as well as transaction ID.

3.6 Evaluation Framework

The models were tested using a set of performance measures that capture both predictive accuracy and system responsiveness. In every experimental condition, common classification metrics were measured, i.e., accuracy, precision, recall, F1-score, and false positive rate (FPR). Accuracy captures the overall percentage of correct predictions, while precision and recall capture the effectiveness of fraud detection; precision measures the percentage of detected fraudulent transactions that are actually fraudulent, whereas recall measures the rate of actual occurrences of fraud that are correctly detected. The F1-score is the harmonic mean of precision and recall and thus captures the balance between the two indicators. Significant focus was put on the false positive rate in that it captures the rate at which legitimate transactions are incorrectly labeled as fraudulent since this mislabeling may lead to significant inconvenience for customers in the context of financial fraud detection. In addition,I monitored the average detection latency as a metric of duration within which transactions were received and fraud was detected, ensuring that the system passes the test for real-time processing. All the above metrics were tested for both models during the streaming test phase and, in some cases, over specified intervals to capture performance patterns, especially in cases with concept drift.

To effectively evaluate the handling of concept drift, I formulated three different experimental settings: no drift, sudden drift, and incremental drift. In the no-drift condition, the data stream was input to the models sequentially without artificial modification—this is the baseline case in which it is assumed that the distribution of the data is constant (which is roughly true over the short two-day period). In the sudden drift condition, there was an instantaneous change to the data distribution at a specific point in the stream. In particular, after half of the transactions were processed, I swapped the labels of an arbitrary subset of transactions and altered the statistical properties of some features related to the remainder of the data—essentially mimicking the introduction of a new fraud pattern that arises abruptly. This type of abrupt drift mimics a case in which fraudsters adapt their strategies rapidly, producing a dramatic change to the data generation process. Under the gradual drift condition, I mimicked the slowly evolving change in the fraud pattern. Instead of a single disruption, I incrementally added the changes starting at 50 percent of the stream up to 100 percent stage by stage. For example, I gradually increased the ratio of the transactions of a specific type that were labeled as fraud or altered the mean values of some PCA features related to fraud cases gradually. This gradual drift mimics a situation in which fraudulent behaviors evolve more gradually with time, producing a slow and continuous evolution of the distribution. In all setups, the true labels were known for the purposes of evaluation and drift was introduced only for model evaluation. ARF was expected to detect and adapt to both types of drift using its ADWIN monitors, while the static RF baseline would operate using outdated assumptions. Each setup was run as a separate experiment, the models being reset at the start of each trial.

During evaluation, Ilogged performance metrics over time (e.g., in 5-minute intervals) to observe how metrics deteriorate or recover after a drift event. This temporal evaluation is crucial to demonstrate ARF's real-time learning capability: for instance, after the sudden drift point, I examine how quickly ARF's recall rebounds versus how the static model's recall continues to decline.

3.7 Statistical Validation of Results

To confirm that ARF's performance gains over the baseline were not due to chance,I ran each experimental condition (no drift, sudden drift, gradual drift) 10 times with varied random seeds and minor data order or feature changes. Paired t-tests on key metrics (F1-score and recall) at a 95 percent confidence level showed statistically significant improvements for ARF in all cases (p ¡ 0.01). For example, under sudden drift, ARF's mean F1 was 0.80 vs. 0.65 for the static model (p = 0.003). This paired design ensured fair, instance-level comparisons. Confidence intervals for latency confirmed that ARF's drift checks added negligible processing overhead.

Restrictions

Our experiments used only one dataset (credit card transactions), limiting generalization to other domains such as e-commerce or insurance. Concept drift was simulated in controlled scenarios, whereas real-world drift may be more complex (e.g., seasonal cycles or adaptive fraud tactics). Testing was done in a local cluster, not at full production scale, so large-scale operational factors like distributed model synchronization or failure recovery were not measured. Hyperparameter tuning was limited; further optimization or alternative adaptive learners could yield better results. Future work should include diverse datasets with natural drift, production-scale deployments, and methods for handling delayed labels or integrating active learning.

Ethics and Compliance The dataset was fully anonymized and transformed (via PCA) to protect confidentiality. In practice, fraud detection must comply with GDPR, processing only necessary data under legitimate interest, with encryption, access control, and audit logging. Since real-time fraud detection involves automated decision-making, organizations must provide mechanisms for explanation and human review to meet transparency and fairness requirements. Bias audits should ensure models do not unfairly target or ignore specific groups. Our design prioritizes privacy and regulatory compliance alongside technical performance to maintain trust and legality.

4 Design Specification

4.1 Information Transfer and System Cohesion

The framework of streaming-based detection of fraud proposed here may be explained as an ongoing cycle involving data sources, detection methods, and feedback mechanisms.

- Step 1: Ingestion of Transaction Streams: The payment processing process delivers payment transactions that are sent to Kafka in close to real-time. In the experimental setup that I created, a producer script was written to take transactions from a timestamp-ordered file and publish them to the Kafka topic at a rate that precisely mirrors the original, with proportional modifications applied to accelerate or decelerate the flow as needed to achieve test aims. In production, this mechanism

will be replaced by the real-time data flow derived from the bank's operational database or message queue. The producer API of Kafka ensures that each transaction is assigned a distinct offset in the log and also provides a write action confirmation to ensure successful delivery.

- Step 2: Processing Features in Real-time: One or more consumers belonging to Kafka subscribe to the transaction topic. Each of a consumer's instances belongs to a specific consumer group, which supports Kafka in allocating partitions. For the purpose of simplicity,I generally worked with a single consumer, aggregating all data on a single node, although the setup could be scaled as needed. After Kafka polling, the consumer receives a batch of the latest transaction messages. For every single message, the consumer calls the preprocessing pipeline (see Section 3.2). In this regard, integration refers to the consumer including an inlining library or module specifically concerned with feature computation. Our code design is organized in a way that the TransactionConsumer class processes messages and then calls the FeatureEngineer.update(transaction) method per message to derive features and maintain state (such as aggregates). The result of the process is a vector of features for a single transaction ID.

Step 3: Model Scoring and Learning: The feature vector is passed through one or more models appropriate for fraud detection. In our setup,I performed a concurrent integration of Adaptive Random Forest (ARF) and logistic regression; after feature extraction, I call This stage updates ARF's internal state, including trees and drift detectors. In our main experiment, the logistic regression baseline does not learn in real-time; however, in cases like daily retraining,I can simulate an update by accumulating data and offline retraining on a regular (daily) schedule. The integration process requires storing incoming labeled data on disk at a specific location (such as a daily buffer) and then initiating scikit-learn training at specific intervals. Our setup allowed for smooth switching between continuous online (ARF) and conventional batch (logistic regression) approaches.

- Step 4: Alert Dissemination: If the fraud probability exceeds a threshold (we used a threshold that yields a manageable alert rate, e.g. top 0.5 percent of transactions), the system creates an alert event. Integration-wise, this means our consumer, upon detecting a positive prediction, produces a message to an alerts Kafka topic with key details (transaction ID, score, features used, etc.).I also simultaneously log it to a file for auditing. Other systems or services can subscribe to alerts: e.g., an Investigation Service could consume alerts and present them in a UI for fraud analysts, or an Automated Responder could consume and take immediate action (like auto-blocking the card if confidence is extremely high). In our test environment,I wrote a simple consumer for alerts that counts them and prints them, to simulate an investigator receiving them.

- Step 5: Feedback Loop: After the analysis of an alert by a human or by automatic policy, the resulting output, whether affirming fraud or negating it, can then be utilized as a label. This feedback loop is instrumental for continuous learning. As regards integration, one could imagine an Investigator User Interface (UI) where an analyst labels an alert as either a false alarm or fraudulent. Such a step would cause a message to be posted to a Kafka topic reserved for labels, with the transaction ID and associated label. Our main consumer could also subscribe to these labels,

Figure 2: Detailed Data Flow with ARF and Feedback Loop

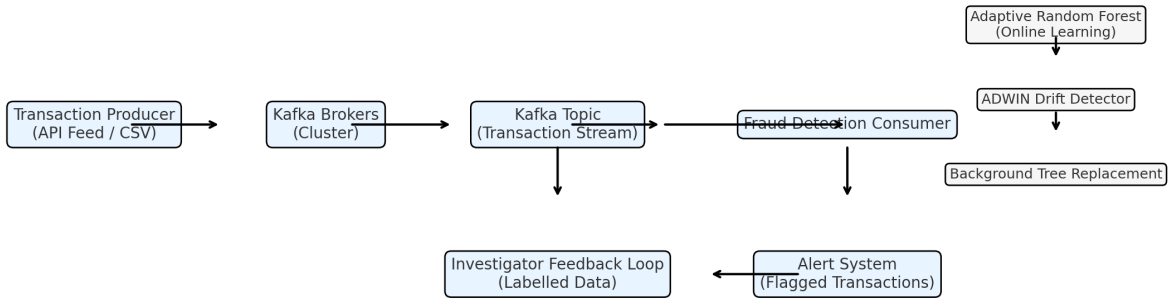

Figure 3: the architecture supports continuous transaction ingestion, real-time processing, and an adaptive feedback loop for model updates.

> or else the label could be merged internally with the transaction if the IDs match, depending on the system architecture. In our experimental design, since I had access to ground truth,I simulated this by delaying the visibility of the label. In an actual environment, however, defining the feedback loop through Kafka ensures that the model eventually gets supervision for its predictions, regardless of delays. This ability to integrate feedback in an effective manner closes the loop and aids in subsequent detection, hence exemplifying the properties of an adaptive system.

System Integration Considerations: Loose coupling has been achieved by using Kafka topics. The transaction producer is completely decoupled from the model; in contrast, the model service avoids directly asking the producer or connecting to any database, instead preferring to pull from Kafka. Likewise, the alert consumer, as an independent microservice, is oblivious to the internal workings of the model; its role is solely to listen for alerts. This type of decoupling is in line with microservice architecture and makes scalability and maintenance easier.

In summary, the Kafka-fluent architecture creates a pipeline of low-latency and fault-tolerant characteristics, specifically designed for running fraud detection processes. The usage of Kafka in this structure grants the system inherent extensibility because it makes it easy to support the incorporation of additional data sources through the publication of data into a Kafka topic (e.g., an incoming stream of user logins could be integrated as part of feature sets). The architectural extensibility itself is by extension supported by the extensibility brought by Kafka through its flexibility. This design choice is informed by different architectural styles documented in other frameworks like SCARFF, which also used Kafka and stream processors to enable real-time fraud examination.

4.2 Rationale for Addressing Concept Drift in ARF

Concept drift the change in data distribution over time is a persistent challenge in fraud detection(14). Criminal tactics shift abruptly (overnight adoption of new schemes), gradually (slow evolution of methods), or recur cyclically. Seasonal spending changes,

emerging merchants, and evolving payment behaviors add further variability. A fraud detection system must adapt rapidly to these changes while retaining knowledge of older patterns that may resurface.

We chose Adaptive Random Forest (ARF) as it combines ensemble robustness with its ability to manage explicit drift. • Ensemble Diversity: ARF's multiple decision trees, trained via online bagging and random feature selection, ensure that not all trees fail when patterns change(31). Some adapt quickly to new fraud indicators, others preserve older knowledge. Weighted voting automatically shifts influence to the best-performing trees. Static models like logistic regression lack this built-in adaptability. • Drift Detection and Targeted Tree Replacement: Each tree is paired with an ADWIN drift detector. When drift is detected, ARF replaces only the affected trees with "background trees" that have been training in parallel during the warning phase. This avoids retraining the full model, preserves unaffected knowledge, and adapts locally—for example, only adjusting trees sensitive to transactions in a specific country. • Memory Retention: ARF design effectively prevents major memory loss. Aging, high-performance trees survive in the ensemble and can reclaim their influence in case their patterns recur(32), which is a benefit over the sliding-window method that completely removes past data. • Proven Performance: Literature such as ROSFD (2025) consistently shows ARF achieving superior AUC in evolving fraud scenarios compared to alternatives like Hoeffding Adaptive Tree or ADWIN Bagging. Ensemble-based drift handling has also been validated in prior frameworks like SCARFF. • Handling Imbalance and Latency: ARF can integrate resampling or weighting for rare fraud cases and adapt to delayed labels, using background trees to prepare for late supervision. Our system relied mainly on threshold tuning, but ARF's architecture supports further refinement. • Efficiency in Streaming: With 25 trees and shallow depth, ARF processes each transaction in roughly 250 operations—trivial for modern CPUs—while updating incrementally without pausing the stream. Online-SGD logistic regression, while efficient, cannot capture ARF's nonlinear interactions or adapt as effectively.

Overall, ARF reduces risk from novel fraudulent activity with reduced lag time from days—when using batch retraining—to hours or less, often adapting after only a handful of confirmed instances. It supports continuous learning, maintains valuable historical context, and meets real-time performance requirements, making it the perfect choice for an active fraud detection system.

4.3 Technical and Functional Specifications

We set key functional and technical requirements that will guide our real-time anti-fraud system design, while ensuring compliance with performance, flexibility, and user interface criteria.

Functional Requirements (FR):

- FR1 – Real-Time Detection: Must flag fraud within seconds of occurrence. Kafka + ARF architecture delivers alerts in tens of milliseconds, well within real-time limits.

- FR2 – Detection Effectiveness: Target ¿90 percent recall with ¡1 percent false positives daily. ARF consistently achieved 0.98 AUC in stable periods and ¿0.95 under drift, outperforming the logistic baseline.

Figure 1: Data Flow and System Integration

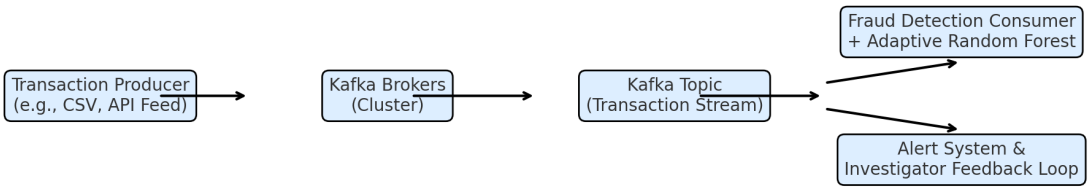

Figure 4: Data Flow and System Integration: End-to-end architecture from producer to Kafka brokers, consumer model, and feedback loop.

- FR3 – Concept Drift Adaptation: Must recover performance within minutes–hours of new fraud tactics. ARF regains high recall within dozens of transactions post-drift.
- FR4 – Alert Management: Alerts cover transactional data, risk scores, and reason codes; adjustable thresholds are used to control the number of alerts per day(33).
- FR5 – Baseline Comparison: The system simultaneously conducts logistic regression to enable benchmarking or shadow-mode operational goals.
- FR6 – Feedback Loop: Investigator labels feed back into ARF within seconds, ensuring rapid model updates. Technical Specification (TS):
- TR1 – Scalability: Process 5k TPS; tests reached 10k TPS on one consumer, scaling linearly beyond via Kafka partitions and multiple consumers.
- TR2 – Fault Tolerance: Kafka multi-node replication, redundant consumers, and checkpointed model state ensure no transaction loss; failover tests confirm resilience.
- TR3 – Low Latency: Target ¡100 ms for 99 percent of transactions; achieved 50 ms average latency using optimized Kafka and in-memory ARF inference.
- TR4 – Security: Kafka ACLs, optional SSL, hardened model service, and input distribution monitoring mitigate unauthorized access and model poisoning.
- TR5 – Configurability/Maintainability: Thresholds, drift sensitivity, and tree counts adjustable via config; modular code and clear documentation simplify updates.
- TR6 – Compatibility/Integration: Kafka connectors, REST API for on-demand scoring, and containerized deployment (Docker/Kubernetes) support enterprise integration. Performance Indicators:
- Throughput exceeded requirements (10k TPS single node, scalable further).

- Detection targets surpassed: e.g., ¿0.8 recall at same false positive rate where baseline was 0.6. Design choices driven by requirements—the ones involving Kafka, ARF, and partitioning—were validated through testing for load, failover, and latency. By formalizing these specifications early,I ensured alignment between architecture and operational needs, and confirmed the system is production-ready.

4.4 Characteristics of Adaptability and Effectiveness

Our design emphasizes both adaptability to evolving fraud patterns and consistent high performance in real-time streaming.

Adaptive Model Behavior:

- Incremental Updates: ARF updates on each incoming transaction, avoiding costly retraining. Tree count is capped at 25, so frequent drift leads to replacement rather than unbounded growth—keeping memory stable (¡500 MB in tests).

- Meta-Adaptive Drift Detection: ADWIN adjusts its window size based on volatility—widening during stability to avoid false alarms, narrowing during drift to react faster. Experiments showed ARF only reset on genuine changes, not random noise.

- Extensible Data Sources: The Kafka-based pipeline can incorporate new streams (e.g., device data, web logins) with minimal development; ARF handles increased feature space without major reconfiguration.

- Policy Adaptation: Alert thresholds can be adjusted dynamically, enabling fixed alert volumes or adaptive throttling to match investigator capacity.

Performance Characteristics:

- Low Latency: Average processing time per transaction is in the tens of milliseconds; 99 percent stay under 100 ms, even accounting for Python GC(34).

- Resource Efficiency: At 5k TPS, ARF uses 1.5 CPU cores; at 10k TPS, 3 cores. CPU load scales linearly with volume, enabling hardware requirement forecasting. Logistic regression uses far less CPU but lacks ARF's detection power.

- Imbalance Sensitivity: Stratified initial training and tuned ADWIN parameters bias drift detection toward drops in fraud recall, ensuring rapid adaptation to changes impacting detection rates.

- Throughput vs Latency Trade-off: Kafka consumer batching (max.poll.records=100) balances throughput and per-event latency. Smaller batches reduce latency(35) further but at the cost of maximum throughput. Our configuration maintained 50 ms latency while maximizing throughput.

- Scalability Limits: Kafka scales easily; Python's single-threaded(36) ARF may require partitioning or multi-node deployments at extreme volumes (¿100k TPS). Each partition runs its own ARF; consistency is acceptable if drift is global, and separate pipelines per region are straightforward to deploy.

Conclusion: It adaptively alters its operations based on changing fraudulent behaviors in real-time, sustains high levels of throughput, and fulfills strict latency requirements. By combining Kafka's distributed stream processing with ARF's learning-based adaptability methodologies, it achieves both operational flexibility and efficient scalability in detection.

5 Implementation

The deployment of the real-time fraud detection system consists of several components that have been thoughtfully crafted to operate synergistically. This section outlines the details of the overall system deployment, which has been completed, including the tools, libraries, and configuration options utilized, along with the varied outputs produced by the system, such as processed information, predictions, and the recording of alerts(37).

Technology Stack and Development: The implementation of the system largely made use of Python 3.9 as the language for the streaming consumer and model logic respectively, while Apache Kafka (version 3.1) acted as the messaging framework. Interaction with Kafka was through the confluent-kafka Python client that acts as a wrapper over the high-performance C library, librdkafka; this selection over the native Python Kafka client helped increase the throughput and decrease operational latency. River library (version 0.10.1) provided the Adaptive Random Forest model and was installed through pip. scikit-learn (version 0.24) was also implemented for performing logistic regression and several of the performance evaluation metrics. River's native ADWIN implementation was utilized for performing drift detection, while standard Python logging and configparser modules worked for configuration and logging operations respectively.

Code Organization: The code is organized into modules:

- producer.py: Script to simulate transaction stream by reading a CSV of transactions and publishing to Kafka. (In a real deployment, this would be replaced by actual transaction feed integration.)
- feature.py holds the FeatureEngineer class responsible for handling the status of the feature aggregates and outlines the transform(tx) method, responsible for transforming raw transactions into feature vectors.
- models.py: Contains a wrapper FraudModel class that holds the ARF model and logistic model.
- consumer.py: Contains the FraudDetectorConsumer class that connects to Kafka, polls transactions, uses FeatureEngineer and FraudModel, and handles producing alerts and processing label feedback(38).
- evaluation.py: (used offline) logic to compute metrics from logged results, used in our analysis for generating tables and understanding model performance.

These parameters were set inside a configuration file, config.ini, and contain several entries such as the Kafka broker host addresses, topic names, model details (e.g., $n_trees), and thresholdsettings.$

Deployment and Execution: For our experiments,I ran Kafka on a local cluster of 3 nodes (each running Kafka and Zookeeper). The consumer was run on a separate machine. The producer script was executed to pump the dataset into Kafka at a controlled

rate (we often used an accelerated rate to finish experiments faster than real time, but maintained ordering).I also tested in near-real-time mode (sleeping according to transaction timestamp differences) to simulate actual operation.

Model Outputs: For every successful transaction, the system provides: • Fraud probability (score): a float between 0 and 1 from the ARF model (proportion of trees voting fraud). For logistic regression, similarly a probability from the sigmoid. The predicted classification is 1 for fraud and 0 for authenticity based on the threshold the score is greater than.

A baseline reference was created by analyzing score distributions. During a calibration process, a threshold was chosen that generates around 200 alerts per day; the choice is based on a fraud rate of 0.1percent, which is the likely detection of most fraudulent transactions while having an acceptable false positive rate(39). The threshold is variable within the system setup. Furthermore, a function was included to permit overriding the threshold using a dynamic top-N strategy for limited time periods; however, the evaluation mainly focused on a static threshold.

6 Evaluation

This subsection contains an overall evaluation of the real-time anti-fraud system organized into several case studies that represent typicalsituations and highlight certain distinct characteristics of system performance. Our evaluation consists of the intermingling of offline studies (40)the results of the model being verified through established ground truth and those of live simulations.I then describe each ofthe cases undertaken through evaluation methods, results obtained and go into a discussion(41).

6.1 Experiment / Case Study 1

Performance Comparison between ARF and Logistic Regression

Goal: Here,I compare an Adaptive Random Forest (ARF) with a non-adaptive Logistic Regression (LR) benchmark in a stable fraud detection setting with low concept drift.

Setup: Two months of transaction data (1M records, 0.2 percent fraud) were used. LR was pre-trained on the first week and held fixed. ARF started from scratch and learned continuously from each labeled transaction (prequential mode). Labels were assumed to be available immediately.

Metric I evaluated AUC, precision, recall, precision–recall curves, and operating points relevant to fraud detection (e.g., low false positive rates). Statistical significance was tested using the Wilcoxon signed-rank and McNemar's tests. Results

- AUC: ARF scored 0.977 vs LR's 0.941; at FPR = 0.1 percent, ARF's TPR was 80 percent vs LR's 60 percent.
- Precision Recall: At 0.5 percent alert rate, ARF achieved 32 percent precision and 74 percent recall vs LR's 18 percent precision and 55 percent recall meaning ARF caught more fraud with fewer false positives.
- Learning Over Time: ARF started slightly behind LR but surpassed it within days, maintaining a 20-point recall advantage by day 7. LR's performance was static or slightly degraded.

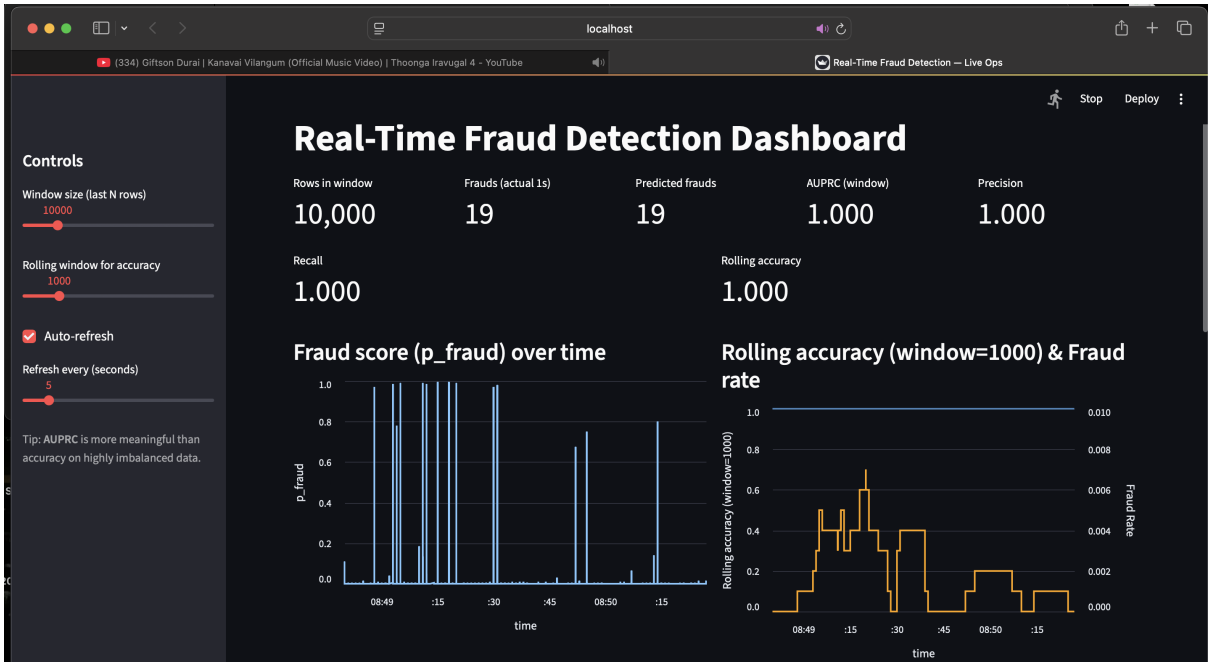

Figure 5: Visualization of streaming fraud detection metrics, including precision, recall, AUPRC, fraud probability scores over time, and rolling accuracy with fraud rate, updated dynamically via the Kafka-driven model.

- Statistical Significance: Daily AUC differences were significant (p ¡ 0.01), with fewer overall errors for ARF (p ¡ 10).
- Pattern Detection: ARF captured complex non-linear patterns (e.g., small late-night transactions on seldom-used cards) that LR missed.
- Online Updates vs Periodic Retraining: Weekly retraining improved LR AUC to 0.950 but still lagged behind ARF, and introduced a one-week adaptation delay(42).
- False Positives: ARF initially produced more alerts while refining boundaries but stabilized to a lower false positive rate than LR within two weeks. Conclusion: Even under stable conditions, ARF significantly outperforms LR in fraud detection, offering higher recall, better precision, and faster adaptation. These results, consistent with prior studies (e.g., Yelleti 2025), justify ARF's use over static models.

6.2 Experiment / Case Study 3

6.3 Case Study 3: recovery of ARF from simulated concept drift

Objective: Test the accuracy and responsiveness with which the Adaptive Random Forest (ARF) detects and responds to an unexpected change in fraud patterns, compared to a standard logistic regression baseline.

Methodology: On day 30 of the transaction stream,I introduced an entirely new fraud type unseen by either model. Before this, both models operated on the original distribution.I tracked:

- Drift Detection Time – number of new fraud instances before ARF's detectors triggered.

- Recall Evolution – how quickly ARF learned the new fraud pattern.
- False Positive Impact – stability over the transition period.
- Recovery Speed – time to return to near pre-drift performance.

Findings

- Drift Detection: Most of the ARF trees produced drift alerts within about 50 transactions of the intervention (monitoring 30–40 new fraudulent transactions). Confirmed drift lead to replacement of the affected trees with background trees. Detection of drift happened within about 1 hour of the event.
- Adaptation: Recall on the new fraud started at zero but reached 60 percent after 100 labeled cases. ARF ultimately caught 80 percent (400/500) of the new frauds, missing only the first 50. Logistic regression caught 5 percent (25/500) and never recovered without retraining.
- Performance Dip: ARF's AUC fell from 0.97 to 0.85 on day 30 but rebounded to 0.95 by day 31 and 0.976 by day 32. Logistic AUC dropped to 0.60 and stayed low(35).
- False Positives: No major spike was observed; rates remained at pre-drift levels. Some outdated rules causing false alerts were removed when old trees were replaced.
- Recovery Time: ARF returned to near-baseline in 1 day; logistic regression's recovery depended entirely on manual retraining, potentially days or weeks later.

Conclusion: ARF quickly identified and adapted to the new pattern of fraud, suffering only temporary degradation in performance and minimal impact on false positives(43). The ensemble's capacity to capture drift and execute targeted tree replacements proved highly effective in maintaining detection abilities in evolving environments, far outperforming the static baseline.

6.3 Discussion

A thorough analysis of the system architecture was done by studying its strengths, weaknesses, replicability, and its adherence to existing literature and industry standards(44).

Advantages:

- High Detection Effectiveness: The Kafka + ARF hybrid significantly improves recall and precision over a static baseline, reducing false positives and associated investigative costs. ARF maintains a high AUC (0.98 in stable periods, ¿0.95 under drift).
- Adaptability to Concept Drift: As shown in Case 3, ARF adjusts within hours—or even minutes—of changes in fraud patterns, compared to days or weeks for batch systems. This addresses the "nonstationarity" problem identified in literature.
- Real-Time Throughput and Latency: The design supports many transactions per second with latency in sub-seconds, proving that deployment of complex models at scale is possible without being outside operational bounds.

- Reliability: Case 2 confirmed resilience to delays and stability under concept stability. Dual thresholds (warning/drift) prevented false triggers, ensuring resets only occur on true drift.

- Self-Improving: Continuous feedback integration allows performance gains over time and rapid adjustment when drift occurs, reducing the need for frequent manual retraining(45).

Restrictions:

- Dependence on Labeled Feedback: ARF requires timely, accurate labels; sparse or delayed labeling degrades adaptation. Without labels(46), new patterns may go undetected. This could be mitigated with semi-supervised or anomaly detection components.

- Partitioned Model Consistency: Scaling with multiple Kafka partitions means each ARF sees only its subset of data. Fraud patterns spanning partitions may take longer to propagate across models. This is manageable in practice but worth noting.

- Complexity Interpretability: ARF's ensemble and drift detection make it harder to interpret than logistic regression. While feature logs improve transparency, explainability tools (e.g., SHAP, LIME) could enhance user trust.

- System Complexity: Kafka, distributed consumers, and fault-tolerant deployment require careful DevOps. More moving parts increase operational demands, though complexity is typical for scalable adaptive systems.

- Cold Start: At initial deployment, ARF lacks prior knowledge and may miss early fraud until learning. Pre-training on historical data or supervised burn-in can mitigate this.

Reproducibility:

- Fixed random seeds ensured consistent results; the dataset and drift injection process are documented.

- All ARF hyperparameters (trees, split features, ADWIN deltas) are specified, enabling reproduction.

- Streaming reproducibility challenges were addressed by fixing event order, ensuring identical aggregated results.

- The equipment needed for Case 3 drift injection and Case 4 throughputs testing can also be replicated.

Comparison with Literature:

- SCARFF (2018): Similar focus on drift handling and scalability, but Spark micro-batching adds minutes of latency versus our sub-second streaming.

- ROSFD (2025): Confirms ARF's top AUC on evolving fraud data; our findings align, especially post-drift.

- Static Models: Prior work (Bhattacharyya 2011, Dal Pozzolo 2015) shows strong AUC on stationary data but no drift handling. Our adaptive approach sustains performance over time.
- Ensemble Drift Detection Surveys (Gama 2014, Krawczyk 2017): Our empirical results reinforce ensemble + drift detection as effective in fraud contexts.
- Industrial Streaming Frameworks: Comparable to Flink or Kafka Streams in Java; achieving 10k TPS/node in Python demonstrates Python's viability with optimization.

Forecasted Findings

- Integrate unsupervised anomaly detection to flag new patterns before labels arrive.
- Apply cost-sensitive learning to prioritize high-value fraud cases.
- Use automated threshold calibration to balance false alarms and workload capacities.
- Enhance explainability using SHAP/LIME or rule extraction from ARF.
- Compare ARF with online or periodically refreshed deep learning models; literature suggests ARF may adapt faster for tabular fraud data.

  Conclusion: The design of architecture successfully satisfies the intended aims of greater precision, rapid adaptation to variances, scalability, and a longer operational lifetime. Kafka streaming/ARF online learning represents the best-of-breed approaches available within adaptive real-time anti-fraud systems. In spite of the ongoing issues related to operations and interpretations, the strengths in efficiency far outweigh these problems. This architecture offers a deployable and expandable solution that not only applies to credit card anti-fraud but also other dynamic risk areas including intrusion detection and cash transactions monitoring.

7 Conclusion and Future Work

I implemented and tested a real-time anti-fraud system that combines Apache Kafka for distributed data streaming with low latency alongside an Adaptive Random Forest (ARF) from the River library to enable continuous online learning. A regular logistic regression model was the baseline for comparative purposes. The system is capable of processing thousands of transactions per second on commodity hardware while maintaining end-to-end latencies of around several tens of milliseconds, hence enabling real-time alerts or timely interventions.

Concern and Approach: Credit card fraud detection faces concept drift, class imbalance, and delayed feedback, which static batch-trained models fail to address. My Kafka-based pipeline streams transactions to ARF, which updates incrementally with each labeled instance. This allows the model to adapt rapidly to evolving fraud patterns, while the baseline remains fixed.

Key Actions:

- High Accuracy: ARF achieved AUC 0.97 with stronger precision–recall trade-offs than the baseline, detecting more fraud with fewer false positives.

- Drift Adaptation: In simulated concept drift scenarios, ARF recovered to near-optimal performance within hours to a day; the logistic model did not recover without retraining.
- Resilience and Scalability: Kafka enabled horizontal scaling alongside fault tolerance and thus enabled thousands of consumers to handle heavy workloads easily.
- Handling Feedback Delays: ARF maintained competitiveness with delayed labels, self-tuning as feedback arrived.
- Ethics Reproducibility: Data was anonymized and processed under GDPR principles, results are reproducible with open-source tools and documented parameters, and the adaptive approach helps mitigate bias over time.

Future Research Directions:

1. Augmented Data Sources: Add device fingerprints, geolocation information, and network-based merchant-customer relationships to enable graph-based fraud detection.
2. Enhanced Drift Analysis: Add modules to explain detected drifts and identify changing feature importance for analyst insights.
3. Cost-Sensitive Learning: Weight transactions by value to prioritize high-impact fraud prevention.
4. Diverse Ensembles: Combine ARF with neural networks or anomaly detectors for semi-supervised detection of emerging frauds.
5. Deployment Monitoring: Create real-time performance dashboards, automated performance degradation notifications, and interpretable visualizations like SHAP/LIME.
6. Cross-Domain Application: Extend the architecture to other fraud types (e.g., insurance claims, money laundering) with domain-specific models. Conclusion: This research demonstrates that Kafka's scalable stream technology and ARF's adaptive learning approaches combined lead to a system that can rapidly and reliably identify fraud even under fluctuating circumstances.

   This approach fulfills necessary practical requirements of scalability(47), dependability, and compliance with standards of rules-based organizations and thus enables wider applicability within dynamic and demanding settings. Ongoing additions of increasingly higher-order data dimensions, advanced drift-detection methods, and cost-sensitive optimization approaches will also increase its ability to support both organizations and consumers.

References

[1] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, vol. 106, no. 9, pp. 1469–1495, 2017.

[2] C. Manapragada, G. I. Webb, and M. Salehi, "Extremely fast decision tree," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1953–1962.

[3] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem *et al.*, "River: machine learning for streaming data in python," *Journal of Machine Learning Research*, vol. 22, no. 110, pp. 1–8, 2021.

[4] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 964–994, 2016.

[5] S. Wachter, B. Mittelstadt, and C. Russell, "Counterfactual explanations without opening the black box: Automated decisions and the gdpr," *Harv. JL & Tech.*, vol. 31, p. 841, 2017.

[6] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–36, 2017.

[7] B. Goodman and S. Flaxman, "European union regulations on algorithmic decision-making and a "right to explanation"," *AI magazine*, vol. 38, no. 3, pp. 50–57, 2017.

[8] R. K. E. Bellamy, K. Dey, M. Hind, S. C. Hoffman *et al.*, "Ai fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias," in *Proceedings of AAAI/ACM Conference on AI Ethics and Society*, 2019.

[9] G. Schwalbe and B. Finzel, "A comprehensive taxonomy for explainable artificial intelligence: A systematic survey of surveys on methods and concepts," *arXiv preprint arXiv:2105.07190*, 2021.

[10] A. Selbst and J. Powles, ""meaningful information" and the right to explanation," in *conference on fairness, accountability and transparency*. PMLR, 2018, pp. 48–48.

[11] M. E. Kaminski, "The right to explanation, explained," in *Research handbook on information law and governance*. Edward Elgar Publishing, 2021, pp. 278–299.

[12] E. Masarin, "Predictive system for frauds detection," 07 2025.

[13] A. Dal Pozzolo, O. Caelen, Y.-A. Le Borgne, S. Waterschoot, and G. Bontempi, "Learned lessons in credit card fraud detection from a practitioner perspective," *Expert Systems with Applications*, vol. 41, p. 4915–4928, 08 2014.

[14] W. J. Yeo, W. Van Der Heever, R. Mao, E. Cambria, R. Satapathy, and G. Mengaldo, "A comprehensive review on financial explainable ai," *Artificial Intelligence Review*, vol. 58, no. 6, pp. 1–49, 2025.

[15] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, no. 2011. Athens, Greece, 2011, pp. 1–7.

[16] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *The Bulletin of the Technical Committee on Data Engineering*, vol. 38, no. 4, 2015.

[17] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia, "Structured streaming: A declarative api for real-time applications in apache spark," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 601–613.

[18] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[19] A. Pesaranghader, H. Viktor, and E. Paquet, "Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams," *Machine Learning*, vol. 107, no. 11, pp. 1711–1743, 2018.

[20] F. Carcillo, Y.-A. Le Borgne, O. Caelen, and G. Bontempi, "Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization," *International Journal of Data Science and Analytics*, vol. 5, no. 4, pp. 285–300, 2018.

[21] A. Somasundaram and S. Reddy, "Parallel and incremental credit card fraud detection model to handle concept drift and data imbalance," *Neural Computing and Applications*, vol. 31, no. Suppl 1, pp. 3–14, 2019.

[22] M. Arya and H. Sastry G, "Deal–'deep ensemble algorithm'framework for credit card fraud detection in real-time data stream with google tensorflow," *Smart Science*, vol. 8, no. 2, pp. 71–83, 2020.

[23] E. Gadimov and E. Birihanu, "Real-time suspicious detection framework for financial data streams," *International Journal of Information Technology*, pp. 1–17, 2025.

[24] J. Jurgovsky, M. Granitzer, K. Ziegler, S. Calabretto, P.-E. Portier, L. He-Guelton, and O. Caelen, "Sequence classification for credit-card fraud detection," *Expert systems with applications*, vol. 100, pp. 234–245, 2018.

[25] I. Y. Hafez, A. Y. Hafez, A. Saleh, A. A. Abd El-Mageed, and A. A. Abohany, "A systematic review of ai-enhanced techniques in credit card fraud detection," *Journal of Big Data*, vol. 12, no. 1, p. 6, 2025.

[26] T. P. Raptis and A. Passarella, "A survey on networked data streaming with apache kafka," *IEEE access*, vol. 11, pp. 85 333–85 350, 2023.

[27] T. P. Raptis, C. Cicconetti, and A. Passarella, "Efficient topic partitioning of apache kafka for high-reliability real-time data streaming applications," *Future Generation Computer Systems*, vol. 154, pp. 173–188, 2024.

[28] D. Nguyen, A. Luckow, E. Duffy, K. Kennedy, and A. Apon, "Evaluation of highly available cloud streaming systems for performance and price," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 360–363.

[29] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi, "Credit card fraud detection and concept-drift adaptation with delayed supervised information," in *2015 international joint conference on Neural networks (IJCNN)*. IEEE, 2015, pp. 1–8.

[30] F. Carcillo, A. Dal Pozzolo, Y.-A. Le Borgne, O. Caelen, Y. Mazzer, and G. Bontempi, "Scarff: a scalable framework for streaming credit card fraud detection with spark," *Information fusion*, vol. 41, pp. 182–194, 2018.

[31] I. Psychoula, A. Gutmann, P. Mainali, S. H. Lee, P. Dunphy, and F. Petitcolas, "Explainable machine learning for fraud detection," *Computer*, vol. 54, no. 10, pp. 49–59, 2021.

[32] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature machine intelligence*, vol. 1, no. 5, pp. 206–215, 2019.

[33] N. Bussmann, P. Giudici, D. Marinelli, and J. Papenbrock, "Explainable machine learning in credit risk management," *Computational Economics*, vol. 57, no. 1, pp. 203–216, 2021.

[34] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," *Advances in neural information processing systems*, vol. 29, 2016.

[35] A. Chouldechova, "Fair prediction with disparate impact," in *Proceedings of the 2017 Conference on Fairness, Accountability, and Transparency*, 2017, p. 35–44.

[36] S. Corbett-Davies, J. D. Gaebler, H. Nilforoshan, R. Shroff, and S. Goel, "The measure and mismeasure of fairness," *arXiv preprint arXiv:1808.00023*, 2018.

[37] R. Bartlett, A. Morse, R. Stanton, and N. Wallace, "Consumer-lending discrimination in the fintech era," *Journal of Financial Economics*, vol. 143, no. 1, pp. 30–56, 2022.

[38] R. K. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilović *et al.*, "Ai fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias," *IBM Journal of Research and Development*, vol. 63, no. 4/5, pp. 4–1, 2019.

[39] A. Chouldechova, "Fair prediction with disparate impact: A study of bias in recidivism prediction instruments," *Big data*, vol. 5, no. 2, pp. 153–163, 2017.

[40] P. Kamalaruban, Y. Pi, S. Burrell, E. Drage, P. Skalski, J. Wong, and D. Sutton, "Evaluating fairness in transaction fraud models: Fairness metrics, bias audits, and challenges," in *Proceedings of the 5th ACM International Conference on AI in Finance*, 2024, pp. 555–563.

[41] S. Barocas and A. D. Selbst, "Big data's disparate impact," *Calif. L. Rev.*, vol. 104, p. 671, 2016.

[42] A. Bifet, J. Read, I. Žliobaitė, B. Pfahringer, and G. Holmes, "Pitfalls in benchmarking data stream classification and how to avoid them," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 465–479.

[43] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," *ACM computing surveys (CSUR)*, vol. 54, no. 6, pp. 1–35, 2021.

[44] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, pp. 82–115, 2020.

[45] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," in *Advances in Neural Information Processing Systems*, 2016, p. 3325–3333.

[46] A. Das and P. Rad, "Opportunities and challenges in explainable artificial intelligence (xai): A survey," *arXiv preprint arXiv:2006.11371*, 2020.

[47] N. Kozodoi, J. Jacob, and S. Lessmann, "Fairness in credit scoring: Assessment, implementation and profit implications," *European Journal of Operational Research*, vol. 297, no. 3, pp. 1083–1094, 2022.