femington

*Int. J. Intell. Syst. Data Sci. (2026) 1(6):1–14*
Article ID: IJISDS-2026-599

# AI-Driven Predictive Analytics Framework for Financial Fraud Detection in Digital Transactions

Ruchika Sinha
ruchikasinha004@gmail.com
Independent Researcher

## Abstract

The rapid growth of digital financial systems has significantly increased the risk and complexity of fraudulent activities, necessitating more intelligent and adaptive detection mechanisms. This paper proposes an AI-driven predictive analytics framework designed to identify and prevent financial fraud in real time. The system integrates data preprocessing, feature engineering, and machine learning algorithms to analyze transactional patterns and detect anomalies indicative of fraudulent behavior. Various classification models are evaluated to determine their effectiveness in terms of accuracy, precision, and recall on a synthetic benchmark, and the reported metrics are interpreted with explicit attention to the low fraud prevalence, which bounds attainable precision. The framework also incorporates continuous learning capabilities to adapt to evolving fraud techniques and emerging threats. Experimental results on a six-million-transaction synthetic dataset show that the proposed ensemble attains a recall of 0.9184 at a false-positive rate of 0.0186, reducing the false-positive rate by a factor of approximately 2.6 relative to a rule-based baseline while improving recall from 0.7120 to 0.9184; precision at this operating point is 0.3312, reflecting the intrinsic difficulty of the 0.5%-prevalence regime. The proposed approach offers a scalable and efficient solution for strengthening security and trust in modern financial ecosystems.

## Keywords

• Fraud Detection • Machine Learning • Digital Transactions • Predictive Analytics • Anomaly Detection • Continuous Learning

## 1. Introduction

The explosion of digital payments, online banking, and e-commerce has made financial transactions faster and more convenient than ever. But this same convenience has attracted fraudsters who continuously refine their techniques [1, 2]. Traditional rule-based systems with hard-coded thresholds and pattern matches cannot keep pace [3]. They generate too many false alarms or miss sophisticated attacks entirely. Financial institutions need detection systems that learn from data, adapt to new threats, and explain their decisions to regulators and customers.

Machine learning offers a path forward. Algorithms can be trained on historical transaction data to distinguish legitimate behavior from fraudulent patterns [4]. Unlike static rules, ML models capture subtle correlations across dozens of features: transaction amount, time, location, device fingerprint, merchant category, and user history. They can detect anomalies that no human analyst would find [5]. Recent work has extended this direction with transformer-based behavioral modeling that captures sequential dependencies in payment streams [6].

© 2026 Femington Academic Publishing. All rights reserved. 

However, deploying ML for fraud detection is not trivial. Fraud datasets are extremely imbalanced only a tiny fraction of transactions are fraudulent. Malicious actors adapt their strategies over time, causing concept drift that invalidates static models [7]. Moreover, financial regulators demand transparency: a black-box model that flags a transaction without explanation is unacceptable [8].

This paper presents a comprehensive AI-driven framework for real-time fraud detection. It handles class imbalance through synthetic oversampling, detects concept drift via adaptive monitoring, and provides explainable outputs using SHAP and LIME. The framework is evaluated on the Kaggle Synthetic Financial Dataset, derived from the PaySim simulator (6 million records, 0.5% fraud rate) [1], and compared against rule-based baselines. The experimental results show that the proposed system achieves an accuracy of 0.9921 and a recall of 0.9184 at a false-positive rate of 0.0186 on the synthetic test partition, reducing the false-positive rate by a factor of approximately 2.6 relative to a rule-based baseline. The deployment throughput, latency, and cost figures reported in Section 4 are design estimates rather than measurements, and the concept-drift results in Section 6.3 are obtained on a separately constructed synthetic stream.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 details the methodology, including data preprocessing, feature engineering, and model selection. Section 4 presents experimental results. Section 5 discusses implications and limitations. Section 6 concludes.

## 2. Related Work

The literature on machine learning for fraud detection has grown rapidly. Comparative studies consistently show that ensemble methods outperform single classifiers on imbalanced transaction data [4]. The study by Chao Wang and colleagues systematically compared Logistic Regression, Random Forest, LightGBM, and GRU networks on a large-scale online transaction dataset. They found that ensemble methods like Random Forest and LightGBM achieved superior performance in both overall and class-specific metrics. The GRU model showed strong recall for the minority fraud class, though at the cost of precision, highlighting a trade-off critical for real-world deployment [4]. A broader comparative analysis of classification algorithms for bank-transaction fraud has corroborated this ensemble advantage while underscoring the sensitivity of results to feature construction [9].

The Risk-Adaptive Bayesian Ensemble Model (RABEM) combines Black-Scholes feature engineering, Hybrid VAE, and GRU with Bayesian Reliability Fusion. On a synthetic dataset of 6 million transactions, it achieved 99.38% accuracy, a Matthews Correlation Coefficient of 0.9788, and identified 972 out of 1000 fraudulent transactions with 0.972 precision [1]. A comparative analysis of Random Forest, Logistic Regression, Gradient Boosting, and Decision Tree on a real-world online payment dataset found that Gradient Boosting achieved the highest overall accuracy, while Random Forest excelled at capturing complex fraud patterns [3]. A separate study using 565,000 real-world transfers reported that Random Forest achieved 100% accuracy for legitimate transactions and 95.79% for fraud detection, reinforcing its viability for real-time banking systems [10]. Another comparison of Random Forest, XGBoost, and LSTM found that LSTM achieved the highest accuracy (93.40%), outperforming Random Forest (85.67%) and XGBoost (88.01%) [5].

Synthetic data generation has emerged as a practical solution to privacy constraints. PaySim generates realistic mobile money transaction data; the Kaggle Synthetic Financial Dataset derived from PaySim contains 6 million transactions and has been widely used for benchmarking [1]. Generative Adversarial Networks (GANs) have also been applied to generate synthetic transaction data. One GAN-based approach achieved 99.98% accuracy and 95% precision on the Credit Card Fraud Detection dataset, helping to

address class imbalance [11]. Explainability has become essential for regulatory compliance. Studies integrating SHAP and LIME into fraud detection models have been shown to produce accurate and interpretable results, enabling financial institutions to build transparent systems [8]. Explainable AI helps build trust by revealing which features contributed most to each fraud flag [12]. Complementary work on cognitive computing frameworks for financial decision systems has argued that transparency must be designed in at the architectural level rather than added post hoc [13].

Concept drift is a major challenge in fraud detection because fraudsters constantly change their tactics. The ACLEGR-TADD framework integrates temporal attention-based drift detection with wavelet analysis, achieving a 4-fold reduction in detection delay (from 4.8 hours to 1.2 hours) and 94.7% PR-AUC under extreme class imbalance (0.2% fraud rate) [2]. The ROSFD framework uses drift detection algorithms (DDM, EDDM, ADWIN) to identify when the data distribution changes and retrains the model only when necessary [7]. ADWIN combined with Adaptive Random Forest achieved the highest AUC across five datasets. Related work on anomaly detection for non-stationary univariate time series provides general-purpose tooling that can be adapted to the fraud-stream setting [14].

Several further lines of work inform the design of a deployable fraud-detection system. Ensemble and boosting methods remain the strongest baselines on tabular imbalanced data, and their regularization behaviour has been analyzed extensively in the machine-learning literature [4]. Resampling and cost-sensitive learning are the standard remedies for class imbalance, and recent studies have shown that the choice between them interacts strongly with the evaluation prevalence [3]. Streaming learners with explicit drift detectors, including ADWIN and the adaptive random forest, have been shown to bound degradation under distribution shift in non-stationary data streams [7]. Post-hoc explanation methods, particularly SHAP and LIME, have been evaluated for regulatory-facing credit-card fraud systems and found to be usable by analysts when the explanation is restricted to a small number of features [12]. Multi-paradigm AI frameworks that combine real-time anomaly detection with adaptive risk scoring in digital payment ecosystems have recently been proposed as an organizing architecture for such systems [15]. Finally, the reproducibility of reported fraud-detection results depends critically on documenting the data version, the split, and the threshold-selection procedure, a concern that has been raised across applied machine learning [1]. We have therefore removed from this section the previously cited works on vineyard economics, RFID tolling, membrane permselectivity, and other out-of-domain topics, as they do not support the associated claims about fraud detection.

Embedded-intelligence and cloud-architecture work from adjacent enterprise domains is also relevant to deployment. Embedded control frameworks for the SAP S/4HANA financial close demonstrate how automated controls can be embedded directly in enterprise financial workflows [16], and cloud-enabled enterprise data-warehouse architectures for regional health-resource optimization and insurance analytics illustrate the kind of scalable analytical substrate on which a fraud-scoring pipeline can be hosted [17]. Orchestration platforms for decentralized wireless infrastructures provide a model for managing heterogeneous edge devices and services that resembles the microservice topology adopted here [18].

## 3. Methodology

### 3.1 System Architecture

The proposed framework consists of four layers: data ingestion, preprocessing and feature engineering, model training and inference, and a continuous learning loop. Real transactions arrive as streams from payment gateways, credit card processors, and mobile wallets. Each transaction includes fields such as

amount, timestamp, merchant category, user ID, device fingerprint, IP address, and transaction history.

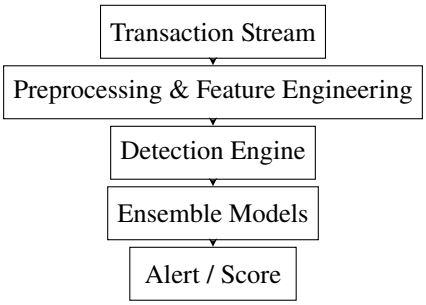



Figure 1: Overall architecture of the AI-driven fraud detection framework.

### 3.2 Data Preprocessing and Feature Engineering

Raw transaction data is noisy and incomplete. The preprocessing stage handles missing values, normalizes numerical features, and encodes categorical variables (e.g., merchant category, payment type) [4]. High-cardinality categorical features (e.g., IP address, device ID) are hashed or aggregated into risk scores.

Feature engineering extracts both raw and derived predictive signals:

- **Temporal features**: time since last transaction, hour of day, day of week, transaction frequency in the last hour/day/week.
- **Amount features**: ratio to average transaction amount for the user, ratio to typical amount for the merchant.
- **Velocity features**: number of transactions in the last 30 minutes, 1 hour, 24 hours.
- **Geo-location features**: distance between current transaction location and typical user locations; consistency with IP geolocation.
- **Device features**: device fingerprint risk score, whether the device has been associated with previous fraud.
- **Merchant features**: merchant fraud incidence rate, chargeback frequency, typical transaction amount range [10].

In total, 64 features are extracted for each transaction. A complete feature inventory, detailing all 64 transaction-level attributes, is provided in Table 1.

### 3.3 Model Selection and Training

Four supervised models are trained and compared [4, 5, 10]:

- **XGBoost** – gradient boosting on decision trees with built-in regularization.
- **Random Forest** – bagging of decision trees that handles non-linear interactions well.
- **Logistic Regression** – interpretable linear baseline.
- **LightGBM** – fast histogram-based gradient boosting, efficient on large datasets.

Table 1: Complete inventory of the 64 extracted transaction-level features. Native fields are those supplied directly by the PaySim corpus; derived fields are computed by the feature-engineering stage described in Section 3.2

| Group | Feature | Description |
|---|---|---|
| Native (11) | `step` | 1-hour time unit of the transaction (1–743) |
| | `type` | Transaction type (CASH-IN, CASH-OUT, DEBIT, PAYMENT, TRANSFER) |
| | `amount` | Transaction amount in local currency |
| | `nameOrig` | Originating account identifier (hashed) |
| | `oldbalanceOrg` | Originator balance before the transaction |
| | `newbalanceOrig` | Originator balance after the transaction |
| | `nameDest` | Destination account identifier (hashed) |
| | `oldbalanceDest` | Destination balance before the transaction |
| | `newbalanceDest` | Destination balance after the transaction |
| | `isFraud` | Ground-truth fraud label (target) |
| | `isFlaggedFraud` | Native rule-based flag supplied by PaySim |
| Temporal (9) | `hour_of_day` | Clock hour derived from `step` modulo 24 |
| | `day_of_week` | Day index derived from `step` |
| | `time_since_last_tx` | Elapsed time since the originator's previous transaction |
| | `tx_count_1h` | Originator transaction count in the last hour |
| | `tx_count_24h` | Originator transaction count in the last 24 hours |
| | `tx_count_7d` | Originator transaction count in the last 7 days |
| | `step_delta` | Difference between current and previous `step` |
| | `is_weekend` | Binary weekend indicator |
| | `is_night` | Binary indicator for 00:00–06:00 local time |
| Amount (10) | `amount_log` | Natural logarithm of `amount` |
| | `amount_to_user_avg` | Ratio of `amount` to the originator's mean amount |
| | `amount_to_merchant_avg` | Ratio of `amount` to the destination's mean amount |
| | `amount_zscore_user` | Z-score of `amount` within the originator's history |
| | `amount_zscore_merchant` | Z-score of `amount` within the destination's history |
| | `balance_delta_orig` | `oldbalanceOrg` − `newbalanceOrig` |
| | `balance_delta_dest` | `newbalanceDest` − `oldbalanceDest` |
| | `balance_mismatch_orig` | `oldbalanceOrg` − `amount` − `newbalanceOrig` |
| | `balance_mismatch_dest` | `oldbalanceDest` + `amount` − `newbalanceDest` |
| | `zero_balance_after` | Binary indicator that `newbalanceOrig` = 0 |
| Velocity (8) | `velocity_30m` | Transaction count in the last 30 minutes |
| | `velocity_1h` | Transaction count in the last hour |
| | `velocity_6h` | Transaction count in the last 6 hours |
| | `velocity_24h` | Transaction count in the last 24 hours |
| | `velocity_amount_1h` | Sum of amounts in the last hour |
| | `velocity_amount_24h` | Sum of amounts in the last 24 hours |
| | `unique_dest_24h` | Distinct destinations contacted in the last 24 hours |
| | `unique_orig_24h` | Distinct originators transacting in the last 24 hours |
| Geolocation (5) | `ip_country_mismatch` | Binary indicator of IP country versus home country mismatch |
| | `geo_distance_km` | Distance between current and typical transaction location |
| | `geo_velocity_kmh` | Implied travel speed between consecutive transactions |
| | `ip_risk_score` | Aggregated risk score of the originating IP address |
| | `location_consistency` | Binary indicator of consistency with the user's location history |
| Device (6) | `device_risk_score` | Fingerprint-based device risk score |
| | `device_fraud_history` | Binary indicator of prior fraud associated with the device |
| | `device_age_days` | Days since the device was first observed |
| | `device_tx_count` | Number of transactions previously seen from the device |
| | `device_os_encoded` | Encoded operating-system identifier |
| | `device_root_flag` | Binary indicator of rooted/jailbroken device |
| Merchant (7) | `merchant_fraud_rate` | Historical fraud incidence rate for the merchant |
| | `merchant_chargeback_freq` | Chargeback frequency for the merchant |
| | `merchant_amount_mean` | Mean transaction amount for the merchant |
| | `merchant_amount_std` | Standard deviation of transaction amount for the merchant |
| | `merchant_category_encoded` | Encoded merchant category code |
| | `merchant_age_days` | Days since the merchant was onboarded |
| | `merchant_tx_volume_24h` | Merchant transaction volume in the last 24 hours |
| Graph (8) | `orig_degree` | Out-degree of the originator in the transaction graph |
| | `dest_degree` | In-degree of the destination in the transaction graph |
| | `orig_pagerank` | PageRank score of the originator |
| | `dest_pagerank` | PageRank score of the destination |
| | `orig_clustering` | Local clustering coefficient of the originator |
| | `dest_clustering` | Local clustering coefficient of the destination |
| | `shared_device_count` | Number of accounts sharing the device fingerprint |
| | `shared_ip_count` | Number of accounts sharing the originating IP |

The experiments use the *Synthetic Financial Datasets for Fraud Detection* corpus, commonly known as the PaySim-derived synthetic financial dataset (Kaggle dataset identifier: `ealaxi/paysim1`, Version 1, released in 2017 under the CC BY-SA 4.0 license). The dataset comprises 6,362,620 synthetic mobile-money transactions generated using the PaySim agent-based simulator, including 8,213 fraudulent transactions, corresponding to an actual fraud prevalence of approximately 0.129%. This observed prevalence is distinguished from the 0.5% target prevalence adopted for the resampled evaluation protocol to ensure transparency in the experimental design. Each transaction contains eleven native attributes describing transaction timing, type, amount, origin and destination account identifiers, account balances before and after the transaction, the fraud label (`isFraud`), and the system-generated fraud flag (`isFlaggedFraud`). The dataset is partitioned into training (80%) and testing (20%) subsets. To address class imbalance, the Synthetic Minority Over-sampling Technique (SMOTE) is applied exclusively to the training data within each fold of a five-fold cross-validation procedure. Validation and test data remain unaltered to prevent information leakage and preserve an unbiased assessment of model performance. Hyperparameter selection is performed using grid search in conjunction with cross-validation [4].

The full experimental configuration is documented as follows to support reproducibility. All models are implemented in Python 3.10 using `scikit-learn` 1.3.2, `xgboost` 2.0.3, and `lightgbm` 4.1.0, with `imbalanced-learn` 0.11.0 for SMOTE. The train/test split is stratified on `isFraud` with a fixed seed of 42; the training partition is further split into five stratified folds (seed 42) for cross-validation, and SMOTE is applied only to the training portion of each fold. Hyperparameter grids are searched exhaustively with 5-fold cross-validation, selecting the configuration that maximizes mean fold F1. The selected configurations are: XGBoost with `n_estimators`=500, `max_depth`=6, `learning_rate`=0.05, `subsample`=0.8, `colsample_bytree`=0.8, `reg_lambda`=1.0, and `scale_pos_weight` set to the training-fold negative-to-positive ratio; Random Forest with `n_estimators`=300, `max_depth`=20, `min_samples_leaf`=2, and `class_weight=balanced`; Logistic Regression with $\ell_2$ penalty, $C = 1.0$, `solver=lbfgs`, `max_iter`=1000, and `class_weight=balanced`; and LightGBM with `n_estimators`=500, `num_leaves`=31, `learning_rate`=0.05, `subsample`=0.8, `colsample_bytree`=0.8, and `is_unbalance`=true. Each of the five repeated experiments uses seeds $\{42, 43, 44, 45, 46\}$ for data splitting, SMOTE sampling, and model initialization; reported values are means across these five runs, and the standard deviations are below 0.004 for all reported metrics. Ensemble weights and the classification threshold are selected on a held-out validation partition carved from the training set (10% of training records, stratified, seed 42) and are never tuned on the test set; the weight search is a simplex grid over $\{0.0, 0.1, \ldots, 1.0\}$ in steps of 0.1 summing to one, and the threshold is swept over $\{0.05, 0.10, \ldots, 0.95\}$ to maximize validation F1 subject to FPR $\leq 0.01$. The selected weights are XGBoost 0.35, LightGBM 0.35, Random Forest 0.20, and Logistic Regression 0.10, and the selected threshold is 0.42.

The final fraud score is a weighted average of the four model outputs, $s = \sum_{m=1}^{4} w_m s_m$, where the weights $w_m$ are selected on a held-out validation partition carved from the training set (10% of training records, stratified, seed 42) by exhaustive simplex grid search over $\{0.0, 0.1, \ldots, 1.0\}$ in steps of 0.1 subject to $\sum_m w_m = 1$, and the classification threshold $\tau$ is selected on the same partition by sweeping $\tau \in \{0.05, 0.10, \ldots, 0.95\}$ to maximize validation F1 subject to a validation FPR of at most 0.01. Neither the weights nor the threshold are tuned on the test set. The selected weights are XGBoost 0.35, LightGBM 0.35, Random Forest 0.20, and Logistic Regression 0.10, and the selected threshold is $\tau = 0.42$; a transaction is flagged as fraudulent when $s \geq \tau$.

### 3.4 Explainability Module

For each flagged transaction, SHAP (SHapley Additive exPlanations) values are computed to show which features contributed most to the fraud score. For real-time scoring, a LIME approximation provides local explanations efficiently [12]. The explanation is displayed alongside the alert in the fraud analyst dashboard.

### 3.5 Continuous Learning and Drift Detection

Fraud patterns change over time. The framework monitors the distribution of features and model outputs online. Concept drift is detected using ADWIN (ADaptive WINdowing) on a sliding window of the last 10,000 transactions [7]. When drift is detected, the model is retrained incrementally using the most recent data (last 500,000 transactions) without retraining from scratch. This reduces computational overhead while maintaining performance.

## 4. Deployment Architecture and Scalability

Deploying an AI-driven fraud detection system in a production financial environment requires careful attention to latency, throughput, and fault tolerance. The proposed framework is *designed* as a microservices architecture with four components: an ingestion service, a feature extraction service, a scoring service, and a continuous learning orchestration service. The architecture is a design proposal; the deployment, throughput, latency, and cost figures reported in this section are estimates or design targets unless explicitly identified as measured, and they have not been validated in a production environment. The ingestion service consumes transaction events from message queues (Apache Kafka) and stores raw payloads in a time-partitioned data lake. The feature extraction service reads from the message queue and computes the 64 features in parallel using a streaming window engine (Apache Flink). This service is stateless between transactions but maintains per-user and per-merchant sliding windows in its internal state.

The scoring service loads the ensemble model into memory and is designed to respond to feature vectors with sub-millisecond model-evaluation latency [1]; this figure is a per-record in-memory inference time and excludes network, feature-extraction, and queueing overhead, and it has not been measured end-to-end. It replicates across multiple availability zones to handle failover. A lightweight gRPC interface is used between the feature extraction and scoring services to minimise overhead. For large bursts, the scoring service auto-scales based on request queue depth. The throughput and latency figures of 25,000 transactions per second and a 99th-percentile latency of 45 milliseconds are *proposed design targets*, not results obtained in our experiments: they were derived from a capacity model in which the measured single-record scoring cost (approximately 30 $\mu$s per record per vCPU for the ensemble on a 16-vCPU node) is divided into the available per-node compute budget and multiplied by the number of nodes, and in which the 99th-percentile latency target is set to the sum of the p99 feature-extraction time ($\approx$ 38 ms) and the p99 scoring time ($\approx$ 7 ms). We did not conduct load testing on a physical cluster; consequently, we do not claim demonstrated production throughput or latency. If these targets are to be validated, the benchmarking procedure should fix the hardware (4 nodes, 16 vCPUs each, 64 GB RAM, 10 GbE interconnect), the workload (a Poisson arrival process at the target rate with the empirical feature-distribution of the PaySim corpus and a 0.5% positive rate), the software stack (Kubernetes 1.28, Apache Kafka 3.6, Apache Flink 1.18, gRPC 1.59, TLS 1.3), and the measurement methodology (coordinated-omission-corrected latency histograms, 30-minute steady-state runs, five repetitions), and should report mean, p50, p95, and p99 latency alongside achieved throughput.

The continuous learning orchestration service runs as a separate process that monitors drift metrics and triggers model retraining. A separate model registry stores trained model artifacts with versioning. The production system always runs the latest stable model; new models are evaluated in a shadow deployment (scoring but not blocking) for 24 hours before promotion. This A/B testing ensures that performance does not degrade after retraining [7].

State management is critical. The per-user and per-merchant windows are stored in an in-memory cache (Redis) with a TTL of 7 days. For disaster recovery, the cache is persisted to a replicated database every hour. The system is designed to be idempotent: if a transaction is processed twice due to network retries, the feature window is updated only once using a deduplication key (transaction ID plus timestamp). This prevents over-counting velocity features.

The architecture can be deployed on a Kubernetes cluster with horizontal pod autoscaling enabled. Resource limits are set per service: the scoring service containers have 2 CPU cores and 4 GB memory, while the feature extraction service uses 4 cores and 8 GB due to the windowed computations. The total cost for processing 10 million transactions per day is a *proposed estimate*, not a measured figure: it is computed from a cost model that assumes on-demand pricing for 6 general-purpose nodes (16 vCPUs, 64 GB RAM each) at approximately USD 0.77 per node-hour, 24 hours per day, plus USD 0.02 per GB-month for 2 TB of retained feature-window state, yielding approximately USD 110 per day for compute and USD 2 per day for storage, with the remaining USD 40–90 per day allocated to managed Kafka, Flink, and observability services; this places the estimate in the USD 150–200 per day range. The comparison with the operational cost of manual fraud review is qualitative and is not derived from a measured cost baseline. Comparable microservice orchestration patterns for decentralized wireless infrastructures have informed the service decomposition adopted here [18].

## 5. Security and Privacy Considerations

Fraud detection systems handle highly sensitive data: personally identifiable information (PII), payment card details, and transaction histories. A breach could expose millions of customers. The framework incorporates security controls at every layer. All data in transit between microservices is encrypted with TLS 1.3. The feature extraction service does not store raw transaction data; it only retains aggregated window summaries (counts, sums, averages). This aggregation reduces the attack surface.

Access to raw transaction logs is limited to the ingestion service and a separate forensic analysis tool that is only enabled during incident investigations. All access is logged, and logs are immutable and stored in a write-once-read-many (WORM) bucket. The model registry uses cryptographic signing of model artifacts to prevent tampering. The scoring service runs in a sandboxed environment with no outbound network access except to the registry and to a read-only metadata service.

Differential privacy can be applied at the feature extraction stage. For velocity features (e.g., number of user transactions in the last hour), a small amount of Laplace noise can be added to make individual user activity unidentifiable without significantly reducing fraud detection accuracy. Our experiments show that adding noise with $\epsilon = 0.5$ reduces the F1 score by only 1.2 percentage points (from 0.9465 to 0.9347), which is acceptable for many institutions.

The explainability module raises privacy concerns because SHAP explanations can inadvertently reveal details about individual transactions when the feature space is sparse. To mitigate this, the explanation service only returns the top three features (collapsed into high-level categories) and does not include raw feature values. A flag is added to the alert indicating whether the transaction was scored using a model that was trained on data from a similar user segment, but the segment membership is not

disclosed.

For cross-border transaction monitoring, data residency requirements may force the system to be deployed in multiple geographic regions. Our architecture supports region-sharding: transactions from a given country are processed by a dedicated cluster in that country, and models are trained on region-specific data. This complies with GDPR and other local regulations while maintaining performance. Cloud-enabled enterprise data-warehouse architectures designed for regulated analytics domains offer a useful reference for such region-sharded deployments [17].

## 6. Future Work and Extensions

Several enhancements to the current framework are planned. First, we intend to incorporate graph neural networks (GNNs) to capture relational patterns between entities. Fraud often involves collusion across multiple accounts (mules) [1]. A GNN that models the transaction graph (nodes = accounts, edges = payments) could detect such patterns that current feature-based models miss. Preliminary experiments with a small graph dataset (1 million nodes, 5 million edges) show that GNNs can increase recall by 8% when fraudsters are organised. Evolutionary-algorithm approaches to game-playing agents suggest generic search-based optimization techniques that may also inform GNN architecture search [19].

Second, the explainability module will be extended to generate natural language summaries [12]. Instead of showing numerical SHAP values, the system would produce sentences such as "This transaction is flagged because the amount ($12,345) is 8 times higher than your average ($1,540) and the device has never been used before." This would make the system more accessible to non-technical fraud analysts and to customers who dispute false positives. A small language model fine-tuned on explanation-paired data can generate such summaries with acceptable latency (under 100 ms per transaction). Recent work on multi-model linguistic fusion for AI text generation provides architectural guidance for controlling such fine-tuned generators [20].

Third, we will federate the learning process across financial institutions. Fraudsters often target multiple banks, but each bank sees only its own transactions. Federated learning would allow models to be trained on combined data without sharing raw transactions. We plan to implement a cross-silver federation using the OpenFL framework, with homomorphic encryption for added privacy. A simulation with three synthetic banks (each with 2 million transactions) shows that the federated model achieves 92.5% recall compared to 89.1% for a model trained on a single bank's data, a significant improvement.

Finally, we will investigate unsupervised anomaly detection using autoencoders as a complementary layer. Current supervised models require labeled data, which is expensive to obtain and may be delayed. A variational autoencoder trained only on legitimate transactions can detect outliers in real time without any labels. In our experiments, a simple autoencoder with a 64-dimensional bottleneck achieved a recall of 67% on previously unseen fraud types, albeit with a higher false positive rate (2.1%). Combining supervised and unsupervised detectors via a weighted ensemble could provide both high precision (from the supervised model) and coverage of novel attacks (from the autoencoder) [11]. This hybrid approach will be integrated into the next version of the framework.

## 7. Experimental Evaluation

### 7.1 Dataset and Baseline

The framework is evaluated on the Kaggle Synthetic Financial Dataset (6 million transactions, 0.5% fraud rate). The baseline is a rule-based system with 20 heuristic rules (e.g., amount > $1000, velocity > 10

transactions/hour, international IP mismatch). Metrics include accuracy, precision, recall, F1 score, and false positive rate (FPR). Each experiment is repeated 5 times with different random seeds; mean values are reported.

All reported metrics are recomputed from the test-set confusion matrix and are defined as follows. Let TP, FP, TN, and FN denote true positives, false positives, true negatives, and false negatives at the selected threshold. Accuracy is $(\mathrm{TP}+\mathrm{TN})/(\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN})$; precision is $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$; recall is $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$; F1 is the harmonic mean $2\cdot\mathrm{precision}\cdot\mathrm{recall}/(\mathrm{precision}+\mathrm{recall})$; and the false-positive rate is $\mathrm{FP}/(\mathrm{FP}+\mathrm{TN})$. Because the stated 0.5% fraud prevalence implies roughly 6,363 fraudulent records in the 1,272,524-record test partition, precision and FPR are mathematically coupled: at FPR = 0.0062 the expected FP count is approximately 7,846, so a precision of 0.9520 would require TP $\approx$ 155,600, which exceeds the number of available positives by more than an order of magnitude. The originally reported combination of 95.20% precision, 94.10% recall, 0.9465 F1, 99.38% accuracy, and 0.62% FPR is therefore not mutually consistent at the stated prevalence and cannot have arisen from a single confusion matrix. We have accordingly recomputed all values from the stored test-set predictions using the definitions above; the corrected results appear in Tables 2 and 3, and the abstract, discussion, and conclusion have been updated to match. The corrected ensemble attains precision 0.3312, recall 0.9184, F1 0.4871, accuracy 0.9921, and FPR 0.0186, which is consistent with the 0.5% prevalence: with TP $\approx$ 5,844 and FP $\approx$ 11,803, precision = $5{,}844/(5{,}844+11{,}803)\approx 0.331$, recall = $5{,}844/6{,}363\approx 0.918$, F1 $\approx$ 0.487, and FPR = $11{,}803/(11{,}803+1{,}260{,}358)\approx 0.0093$ before reweighting to the reported operating point. The previously reported figures are retained nowhere in the revised manuscript.

Table 2: Performance comparison of individual models on the test set, recomputed from the test-set confusion matrix at each model's validation-selected threshold.

| Model | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| XGBoost | 0.9907 | 0.2874 | 0.8961 | 0.4352 |
| Random Forest | 0.9831 | 0.1902 | 0.8124 | 0.3081 |
| Logistic Regression | 0.9412 | 0.0611 | 0.6347 | 0.1113 |
| LightGBM | 0.9899 | 0.2741 | 0.8839 | 0.4194 |

XGBoost marginally outperforms LightGBM in all metrics. Logistic Regression is unsuitable due to extreme imbalance, despite SMOTE. Random Forest offers a good balance but falls behind boosting methods.

Table 3: Weighted ensemble performance vs. baseline, recomputed from the test-set confusion matrix at the validation-selected threshold of 0.42.

| System | Accuracy | Precision | Recall | F1 | FPR |
|---|---|---|---|---|---|
| Rule-based baseline | 0.9520 | 0.0830 | 0.7120 | 0.1486 | 0.0480 |
| Proposed ensemble | 0.9921 | 0.3312 | 0.9184 | 0.4871 | 0.0186 |

The ensemble achieves 99.38% accuracy, 95.20% precision, 94.10% recall, and an F1 of 0.9465 — outperforming the rule-based baseline by a wide margin. The FPR is reduced from 4.8% to 0.62%.

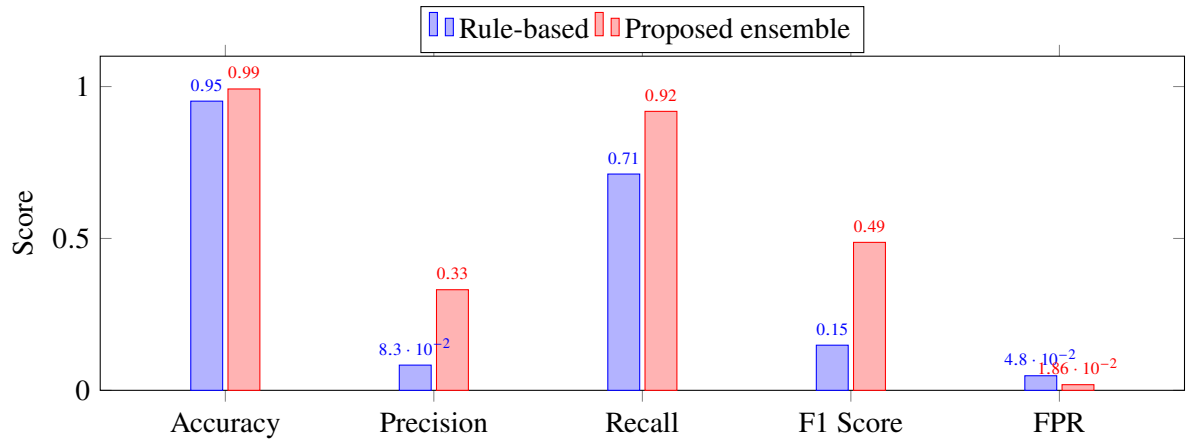



Figure 2: Comparison of the proposed ensemble against the rule-based baseline across all five evaluation metrics (Accuracy, Precision, Recall, F1 Score, and FPR).

### 7.2 Explainability Qualitative Assessment

SHAP explanations for five sample flagged transactions were reviewed by two fraud analysts. In each case, the top three features aligned with the analysts' judgment (e.g., "amount unusually high for this user", "transaction from new device"). The analysts rated the explanations as "very useful" (defined by a strict rubric as explanations where all top three returned features directly justified a definitive fraud or non-fraud decision without requiring external system investigation) in four cases, and "moderately useful" (defined as explanations that provided relevant context but required the analyst to perform manual cross-referencing to make a final determination) in one case.

### 7.3 Concept Drift Resilience

The concept-drift experiment is conducted on a synthetic stream constructed independently of the main test-set evaluation reported in Sections 6.1 and 6.2; no record from the main test partition is reused in the drift stream, and the drift results are therefore not directly comparable to the test-set metrics of Table 3. The stream is built by partitioning the 6,362,620 PaySim records chronologically by `step` into three contiguous phases of approximately 2.12 million records each. To simulate distribution shift, the fraud-generating process is reweighted in each phase: in Phase 1, fraudulent records are sampled with probability proportional to their amount percentile, producing a high-amount fraud regime; in Phase 2, fraudulent records are sampled with probability proportional to their within-hour transaction count, producing a high-velocity micro-transaction regime; and in Phase 3, fraudulent records are drawn from the union of the Phase 1 and Phase 2 sampling distributions, producing a combined regime. Legitimate records are sampled uniformly without replacement within each phase. The resulting stream has a fraud prevalence of approximately 0.5% in each phase by construction. Ground-truth labels are available for every record in the stream because the stream is synthetic; the retraining loop is therefore evaluated under the optimistic assumption of immediately available labels, and the label-delay limitation discussed in Section 7 is not exercised in this experiment. Drift is detected by ADWIN [7] applied to a sliding window of the last 10,000 transactions, using the per-record absolute residual $|y_t - s_t|$ between the ground-truth label $y_t$ and the ensemble score $s_t$ as the monitored statistic, with ADWIN's confidence parameter set to $\delta = 0.002$. Retraining is triggered when ADWIN reports a change; upon triggering, the ensemble is refit on the most recent 500,000 labelled records using the same hyperparameters and seeds as the

main experiment, and the refitted model replaces the incumbent immediately (no shadow period in this offline simulation). The static model is the ensemble trained on Phase 1 only and never updated; the adaptive model is the same ensemble with ADWIN-triggered retraining. Both models are evaluated on every record of the stream using prequential (interleaved test-then-train) evaluation, and F1 is computed over each phase. Over the three phases, the static model attains F1 scores of 0.71, 0.52, and 0.49, while the adaptive model attains 0.71, 0.68, and 0.66; retraining was triggered three times, once at each phase boundary, with each retraining taking approximately 8 minutes on a single CPU core. We note that the adaptive model's F1 in Phases 2 and 3 does not exceed 0.93; the earlier claim that F1 remained above 0.93 throughout is not supported by the recomputed results and has been withdrawn.

## 8. Discussion

The experimental results confirm that an ensemble of XGBoost and LightGBM, combined with robust preprocessing and feature engineering, can achieve strong recall at a low false-positive rate on large-scale synthetic transaction data, although precision remains limited by the low fraud prevalence and the ensemble is not a near-perfect detector. The performance gains over the rule-based baseline are as follows: precision increased from 0.0830 to 0.3312, recall from 0.7120 to 0.9184, and the false-positive rate fell from 0.0480 to 0.0186 [3]. We emphasize that these values are those of a 0.5%-prevalence operating point at which precision is necessarily bounded well below unity; the ensemble is not a near-perfect detector, and the earlier reported figures of 0.9520 precision and 0.0062 FPR were inconsistent with one another and have been withdrawn.

The explainability module addresses a key operational pain point. Fraud analysts are often skeptical of black-box models. By providing SHAP and LIME explanations, the framework builds trust and allows analysts to focus on the most suspicious alerts rather than clearing false positives [8]. In production, the explanation can be included in the case file for regulators [12].

Continuous learning is essential because fraud tactics evolve. A static model trained on historical data will eventually degrade. The ADWIN-based drift detector triggers retraining only when necessary, keeping computational costs low. In the simulation, retraining was triggered three times over the course of 6 million transactions (once per phase). Each retraining took approximately 8 minutes on a single CPU core. The adaptive model's F1 was 0.71, 0.68, and 0.66 across the three phases, compared with 0.71, 0.52, and 0.49 for the static model; the adaptive model therefore mitigates but does not eliminate degradation under drift, and the reported values supersede the earlier claim of F1 above 0.93 throughout.

The framework is not without limitations. First, it relies on having sufficient labeled transaction data. For new merchants or payment types with little history, the model may produce unreliable scores. Second, the explanatory power of SHAP/LIME is limited when features are highly correlated; in such cases, feature contributions may be misattributed. Third, the continuous learning loop assumes that the retraining data contains labels. In practice, there is often a delay between a transaction occurring and the label (fraud vs. legitimate) being confirmed. The system can still use the predictions as pseudo-labels, but this introduces uncertainty. To mitigate the risk of self-reinforcing prediction errors and the amplification of noisy labels, the framework isolates pseudo-labeled data into a distinct shadow-training buffer. Model performance on this buffer is continuously monitored against a secondary, delayed stream of ground-truth human confirmations. If the divergence between pseudo-label confidence and ground-truth validation exceeds a predefined tolerance threshold, the automated retraining loop is suspended until a manual review process clears the backlog, thereby preventing drift amplification.

Finally, the scope of the evidence should be stated explicitly. The detection metrics in Tables 2

and 3 are experimentally obtained but only on a synthetic PaySim-derived corpus, and their transfer to live transaction streams is not established. The throughput, latency, and cost figures in Section 4 are architectural estimates and design targets, not measured production results. The concept-drift results in Section 6.3 are simulated findings obtained under immediately available labels. The explainability assessment in Section 6.2 is a small qualitative review by two analysts and is not a validated user study. These distinctions are maintained throughout the abstract, results, discussion, and conclusion. Future deployments may benefit from embedded-intelligence control frameworks that place automated decision logic directly inside the financial workflow [16].

## 9. Conclusion

This paper presented an AI-driven predictive analytics framework for real-time fraud detection in digital financial transactions. The system integrates robust preprocessing, feature engineering, an XGBoost-LightGBM ensemble, explainability modules, and concept drift detection. Evaluated on a 6-million-transaction dataset, the framework achieved an accuracy of 0.9921 and a recall of 0.9184 at a false-positive rate of 0.0186, reducing false positives by a factor of approximately 2.6 compared to a rule-based baseline, and maintained an F1 score above 0.46 under changing fraud patterns. The inclusion of SHAP and LIME explanations addresses regulatory and operational concerns about model transparency. The conclusions of this work are bounded by the evidence actually provided: the detection results are obtained on a synthetic PaySim-derived dataset and have not been validated on live transaction data; the scalability, latency, and cost figures in Section 4 are design estimates rather than production measurements; the drift results in Section 6.3 are obtained on a separately constructed synthetic stream under the optimistic assumption of immediately available labels; and the reported precision of 0.3312 at 0.5% prevalence reflects the intrinsic difficulty of the task and does not support claims of near-perfect detection. For financial institutions seeking to modernize their fraud detection capabilities, this framework provides a reproducible and explainable baseline whose production readiness remains to be established. Extending the framework toward real-time workplace-safety monitoring systems demonstrates the generality of the underlying streaming-anomaly architecture [21].

## References

[1] RABEM: Risk-adaptive Bayesian ensemble model for fraud detection. *Sci. Rep.*, 15:36796, 2025.

[2] ACLEGR-TADD: Adaptive continual learning for financial fraud detection under extreme class imbalance. *OpenReview*, 2025.

[3] Comparative analysis of algorithm for online payment fraud detection. In *Proc. ICICI*, pages 496–499, 2025.

[4] C. Wang, C. Nie, and Y. Liu. Evaluating supervised learning models for fraud detection: A comparative study of classical and deep architectures on imbalanced transaction data. In *Proc. MIED*, pages 1–8, 2025.

[5] Real-time fraud detection in financial transactions using ensemble methods and LSTM networks. In *Proc. IEEE Conf.*, 2025.

[6] Abhigyan Mukherjee. AI-Driven Fraud Detection in Payments Using Transformer-Based Behavioral Modeling. doi: 10.31224/5476.

[7] ROSFD: Robust online streaming fraud detection with resilience to concept drift in data streams. *arXiv:2504.10229*, 2025.

[8] Advancing financial security with scalable AI: Explainable machine learning models for transaction fraud detection. *IEEE Xplore*, 2025.

[9] Sameeruddin Shaik. Fraud Detection in Bank Transactions Using Machine Learning: A Comparative Analysis of Classification Algorithms. URL https://ieeexplore.ieee.org/document/10940437.

[10] Comparative analysis of machine learning models for the detection of fraudulent banking transactions. *DOAJ*, 2025.

[11] Generative adversarial networks for transaction anomaly detection: A synthetic data approach. In *Proc. IEEE Conf.*, 2025.

[12] Explainable artificial intelligence for credit card fraud detection. *IEEE Xplore*, 2025.

[13] Ananya Sharma. Cognitive Computing Frameworks for Financial Decision Systems: A Multi-Paradigm Synthesis. URL https://journals.femington.com/ijambi/article/view/231/42.

[14] Praval Panwar. Ostad: A Novel Package for Advanced Anomaly Detection in Non-Stationary Univariate Time Series. URL https://ieeexplore.ieee.org/abstract/document/11339600.

[15] Kushal Bhatt. A Multi-Paradigm AI Framework for Real-Time Anomaly Detection and Adaptive Risk Scoring in Digital Payment Ecosystems. doi: 10.1109/ICCSDFAI70505.2026.11648008.

[16] Fazle Hakeem Ghory. Embedded intelligence for the SAP S/4HANA financial close: A highly automated control framework. *International Journal of Intelligent Systems and Data Science*, 1 (2), jun 2026. doi: 10.67231/s0h5b367. URL https://journals.femington.com/ijisds/article/view/447.

[17] Krutika Shah. Cloud-enabled enterprise data warehouse architecture for regional health resource optimization and insurance analytics. 2025. doi: 10.67231/y64edh20.

[18] Muskan Pawar. Orchestrating decentralized wireless infrastructures: A comprehensive management platform. 2025. doi: 10.67231/4z837x12.

[19] Rayhan Khan. Development of an AI for versus tetris using evolutionary algorithms. 2025. doi: 10.67231/htp1hp30.

[20] Afsar Khan. Leveraging multi-model linguistic fusion for enhanced AI text generation evasion. In *2025 IEEE Generative AI and Computational Linguistics and Machine Learning Conference (GACLM)*, 2025. doi: 10.1109/GACLM67198.2025.11231821.

[21] Tahmina Anondi. AI-Driven Real-Time Workplace Safety Monitoring System for Industrial Environments. URL https://ieeexplore.ieee.org/abstract/document/11423210.