iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 14 **Issue:** V **Month of publication:** May 2026

**DOI:** https://doi.org/10.22214/ijraset.2026.82507

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 14 Issue V May 2026- Available at www.ijraset.com*

# Machine Learning-Based Classification System for Digital Payment Fraud Prevention

Prof. Dr. Sachin Takale[1], Martand Tripathi[2], Bhavesh Bachhav[3], Namrata Bade[4], Atharv Shimpi[5]
[1]*Guide, Professor, Dept. of Electronics and Computer Engineering, MIT School of Engineering & Sciences, Pune, India*
[2, 3, 4, 5]*Dept. of Electronics and Computer Engineering, MIT School of Engineering & Sciences, Pune, India*

***Abstract:*** ***This paper presents a comprehensive Online Fraud Detection System developed to identify and prevent suspicious online financial transactions in real time using machine learning and a Java-based enterprise backend. With the exponential rise in digital payments through online banking, UPI, digital wallets, and e-commerce platforms, financial fraud has emerged as a critical challenge for institutions and consumers alike. Traditional rule-based fraud detection systems are insufficient against continuously evolving fraud tactics, necessitating intelligent, adaptive solutions. The proposed system employs a layered architecture comprising a Spring Boot REST API backend, an Apache Kafka message-streaming pipeline, a Hibernate-managed MySQL database, and machine learning models integrated via the Weka and Smile libraries for Java. Algorithms including Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine are evaluated and combined to classify transactions as genuine or fraudulent. The system analyzes transaction attributes such as transaction amount, geographic location, device fingerprint, IP address, time-of-day pattern, and user behavioral history. Upon detecting anomalous activity, real-time alerts are dispatched via the JavaMail API and the Twilio SMS SDK. Experimental results indicate that the Random Forest classifier achieves a classification accuracy of 96.3% with a false positive rate of 1.9% on benchmark datasets. The system is designed for seamless integration with existing banking infrastructure through standardized REST endpoints.***
***Keywords: online fraud detection, machine learning, Java, Spring Boot, Apache Kafka, Random Forest, Weka, transaction analysis, real-time monitoring, financial security.***

## I. INTRODUCTION

The rapid digitization of financial services has transformed the way individuals and businesses conduct transactions. Online banking, Unified Payments Interface (UPI), digital wallets such as Paytm and PhonePe, and e-commerce giants have collectively brought billions of transactions onto digital rails every day. The convenience of these platforms, however, has been accompanied by an equally rapid growth of online fraud. According to the Reserve Bank of India, reported incidents of digital financial fraud have more than doubled in the past three years, resulting in cumulative losses exceeding thousands of crores annually.

Fraudsters exploit a variety of vectors to perpetrate illegal transactions. These include credential theft through phishing emails, SIM card swapping, account takeover using brute-force or credential-stuffing attacks, card-not-present fraud in e-commerce, and synthetic identity fraud where fabricated identities are used to open accounts. The sophistication of these attacks means that static, rule-based detection systems — which rely on a fixed set of human-defined conditions such as transaction amount thresholds or geographic restrictions — are increasingly ineffective. Fraudsters study and circumvent these rules, operating just below threshold values or exploiting timing windows that rules do not cover.

The need for adaptive, intelligent fraud detection systems has therefore become urgent. Machine learning offers a compelling solution: instead of fixed rules, a model learns the statistical distribution of legitimate transactions and identifies deviations from that distribution as anomalies. When new fraud patterns emerge, the model can be retrained on updated data without requiring manual rule engineering. This makes machine learning-based systems fundamentally more resilient to evolving attack strategies.

The system described in this paper addresses the problem of online fraud detection from an engineering perspective, with a complete implementation in Java. Java is chosen as the primary implementation language for several important reasons. First, Java's mature enterprise ecosystem — particularly Spring Boot and Spring Security — provides production-grade infrastructure for building secure, scalable REST APIs that can be integrated directly into banking backend systems. Second, Java's strong type system and compile-time checks reduce the category of runtime errors that could lead to security vulnerabilities in a financial application. Third, Java's multithreading model and the Java Virtual Machine's garbage collection make it well-suited for high-throughput, low-latency transaction processing. Fourth, Apache Kafka, which is used in this system as the real-time transaction streaming pipeline, has a first-class Java client, making the integration natural and performant.

Finally, the Weka machine learning library and the Smile library both provide Java-native implementations of the required classification algorithms, eliminating the need for a separate Python microservice and the associated inter-process communication overhead.

The system operates in four phases: data ingestion via a Kafka consumer pipeline, preprocessing and feature engineering in Java, classification using the trained model, and alert dispatch via the JavaMail API and Twilio. The remainder of this paper is organized as follows. Section II reviews related work. Section III describes the system architecture. Section IV details the methodology. Section V explains the machine learning algorithms. Section VI describes the Java implementation. Section VII presents hardware and software requirements. Section VIII discusses expected results. Section IX covers advantages. Section X covers applications. Section XI discusses future enhancements. Section XII concludes the paper.

## II. RELATED WORK

Research in automated fraud detection has a history spanning over two decades, evolving in parallel with the growth of electronic commerce. Early work focused on neural networks applied to credit card fraud data. More recent work has shifted toward ensemble methods, deep learning, and graph-based approaches.

Sohony et al. [1] applied ensemble learning to credit card fraud detection and demonstrated that combining multiple classifiers through stacking and boosting significantly improved detection rates over single-algorithm approaches. Their ACM study established that model diversity is essential for handling the heterogeneous nature of fraudulent transaction patterns, laying a foundation for the ensemble architecture adopted in this paper.

Xuan et al. [2] explored Random Forest for credit card fraud detection and conducted a comparative analysis against Logistic Regression, SVM, and neural networks on real-world transaction data. Their IEEE ICNSC study found that Random Forest achieved superior precision and recall due to its bagging strategy, which reduces overfitting and handles class imbalance better than single-tree methods. This finding directly motivates the selection of Random Forest as the primary classifier in the proposed system.

Rtayli and Enneya [3] investigated feature selection combined with SVM for credit card fraud detection, demonstrating that carefully selecting the most informative transaction attributes improves model performance while reducing computational cost. Their work in Procedia Manufacturing showed that the combination of Information Gain-based feature ranking with SVM achieved classification accuracy exceeding 92%, validating the feature selection methodology adopted in Section IV-B of this paper.

Zojaji et al. [4] provided a comprehensive survey of credit card fraud detection techniques from both data-oriented and technique-oriented perspectives. Their analysis categorized existing approaches into rule-based, statistical, and machine learning methods, identifying the key challenge of concept drift — the gradual shift in the statistical properties of transaction data as fraud tactics evolve. Their findings motivated the inclusion of an automated weekly retraining pipeline in the proposed system to address concept drift in production deployment.

Benchaji et al. [5] applied genetic algorithms to improve classification on imbalanced fraud datasets, demonstrating that evolutionary feature selection combined with ensemble classifiers yields F1-score improvements of up to 8% over naive oversampling baselines.

Their work reinforces the critical importance of principled class-imbalance handling — addressed in this paper through SMOTE. Misra and Thakur [8] further advanced the field with an AI-driven explainable fraud detection model for digital payments (Frontiers in AI, 2024), combining XGBoost with SHAP explainability to meet emerging regulatory requirements for transparent AI decisions in financial services — a direction considered in the future enhancements of this work.

A notable gap across the reviewed literature is the absence of complete, end-to-end implementations in Java, despite Java being the dominant language in banking and financial services backend systems. Most research implementations use Python with scikit-learn, which is appropriate for experimentation but requires additional engineering to deploy in a Java-based production environment. This paper addresses that gap by presenting a Java-native implementation using Weka and Spring Boot.

## III. SYSTEM ARCHITECTURE

The Online Fraud Detection System is designed around a microservices-inspired layered architecture. Each layer has a clearly defined responsibility and communicates with adjacent layers through well-specified interfaces, enabling independent development, testing, and scaling. The four layers are: the Data Ingestion Layer, the Processing and Feature Engineering Layer, the Detection and Classification Layer, and the Alert and Response Layer. Figure 1 illustrates the overall block diagram.

### A. Data Ingestion Layer

The Data Ingestion Layer is responsible for capturing transaction events from upstream payment systems and making them available to the detection pipeline. In the production architecture, transactions arrive via Apache Kafka topics published by the payment gateway or core banking system. A Spring Boot Kafka consumer application subscribes to the transactions topic and reads each transaction event as a JSON message. Kafka's distributed log architecture guarantees at-least-once delivery and allows the consumer group to scale horizontally by adding consumer instances. For the prototype implementation, transaction data is loaded from a MySQL database table populated from publicly available benchmark datasets including the IEEE-CIS Fraud Detection dataset and the PaySim synthetic mobile money simulation dataset.

### B. Processing and Feature Engineering Layer

Raw transaction records contain fields such as transaction ID, account ID, merchant ID, transaction amount, timestamp, device fingerprint, IP address, and geographic coordinates. The Processing Layer transforms these raw fields into a feature vector suitable for machine learning classification. The feature engineering pipeline, implemented as a Java service class using the Spring Framework, performs the following operations. First, categorical fields such as merchant category code and payment method are encoded using one-hot encoding. Second, continuous fields such as amount are normalized to a zero-mean, unit-variance scale using statistics computed from the training dataset. Third, derived features are computed: time-since-last-transaction, transaction frequency in the past 24 hours, deviation of the current transaction amount from the account's 30-day moving average, and a velocity flag indicating whether more than five transactions have occurred from the same device in the past hour. These derived features capture behavioral context that individual transaction fields cannot express.

### C. Detection and Classification Layer

The Detection Layer receives the feature vector from the Processing Layer and passes it to the trained classification model. The model is loaded from a serialized .model file at application startup and held in memory for the lifetime of the application, avoiding repeated deserialization overhead. The classifier returns a binary prediction — genuine or fraudulent — along with a probability score representing the model's confidence. Transactions with a prediction of fraudulent or with a genuine probability below a configurable threshold (default 0.85) are flagged for further action. The threshold is configurable at runtime through application properties, allowing the risk tolerance of the system to be adjusted without redeployment.

### D. Alert and Response Layer

When a transaction is flagged, the Alert Layer executes one or more response actions depending on the configured risk policy. At the highest risk level (probability below 0.5), the transaction is immediately declined and a block event is written to the database. At the medium risk level (probability between 0.5 and 0.85), the transaction is placed on hold and a step-up authentication challenge is sent to the account holder via SMS using the Twilio Java SDK. At the lower risk level, the transaction is allowed to proceed but flagged for manual review, and a notification email is sent to the fraud operations team using the JavaMail API. All flagged transactions are logged to a fraud_events table in MySQL with full audit fields including the model prediction, probability score, feature vector snapshot, and timestamp, supporting post-incident analysis and model retraining.

## IV. METHODOLOGY

The system follows a structured methodology from dataset preparation through model deployment. The steps are described below.

### A. Dataset Preparation

The primary dataset used for training and evaluation is the IEEE-CIS Fraud Detection dataset, which contains 590,540 transaction records with 433 features and a fraud rate of approximately 3.5%. Class imbalance is addressed using SMOTE, implemented as a preprocessing step in a Java utility class. After oversampling, the training dataset contains a balanced 50:50 split of genuine and fraudulent transactions. The dataset is split into 80% for training and 20% for testing using stratified sampling to preserve the class ratio.

### B. Feature Selection

The 433 raw features in the dataset include many redundant or low-information fields. Feature selection is performed using the Information Gain attribute evaluator provided by Weka, which ranks features by their mutual information with the class label.

The top 40 features are retained, reducing the dimensionality of the problem and improving training speed without significant loss of classification accuracy. Selected features include transaction amount, card type, product category, time deltas between transactions, and several anonymized V-features from the original dataset that represent behavioral biometrics.

### C. Model Training

Model training is performed offline using a standalone Java application that loads the preprocessed dataset from a CSV file, applies the feature selection, instantiates the classifier, trains it, and serializes the resulting model object to disk using Java serialization. The training application uses Weka's Classifier API, which provides a uniform interface across all supported algorithms. For Random Forest specifically, Weka's RandomForest class is configured with 100 trees and a maximum depth of 20, which was found to balance accuracy and inference latency in preliminary experiments.

### D. Model Evaluation

Model performance is evaluated on the held-out test set using the following metrics: accuracy, precision, recall, F1-score, and the area under the Receiver Operating Characteristic curve (AUC-ROC). The AUC-ROC metric is particularly important for fraud detection because it measures the model's ability to discriminate between classes across all possible classification thresholds, independent of the chosen operating point. The confusion matrix is also computed to inspect false positive and false negative rates, as both carry different business costs in a fraud detection context.

## V. MACHINE LEARNING ALGORITHMS

Four supervised classification algorithms are implemented and evaluated. All four are available in Java through the Weka and Smile libraries, enabling a consistent experimental framework. Table I summarizes the comparative performance of the evaluated algorithms on the test dataset.

### A. Logistic Regression

Logistic Regression models the probability of a transaction being fraudulent as a logistic function of a linear combination of the input features. Despite its simplicity, it provides a strong baseline for binary classification tasks and is highly interpretable: the magnitude of the learned coefficients directly indicates the importance of each feature. In Java, Logistic Regression is implemented using Weka's Logistic class. It achieves 88.4% accuracy on the test set, with performance limited by its inability to capture nonlinear relationships between features.

### B. Decision Tree

The Decision Tree algorithm recursively partitions the feature space based on the feature that provides the greatest information gain at each node, producing a tree structure that can be directly interpreted as a set of decision rules. This interpretability is valuable in the fraud detection domain, where risk and compliance teams may need to explain model decisions to auditors. The implementation uses Weka's J48 class, which implements the C4.5 algorithm with post-pruning to reduce overfitting. Decision Tree achieves 91.7% accuracy on the test set.

### C. Random Forest

Random Forest is an ensemble method that trains a large number of Decision Trees, each on a different bootstrap sample of the training data and considering only a random subset of features at each split. The final prediction is the majority vote of all trees for classification tasks. This bagging and feature randomization strategy reduces variance significantly compared to a single Decision Tree while maintaining low bias, making Random Forest robust to outliers and noise in transaction data. The Weka RandomForest class is used, configured with 100 estimators. Random Forest achieves the highest accuracy of 96.3% and the lowest false positive rate of 1.9%, making it the primary classifier deployed in the production pipeline.

### D. Support Vector Machine

The Support Vector Machine finds the hyperplane in the feature space that maximizes the margin between the two classes. For nonlinearly separable data, a kernel function — in this case the Radial Basis Function kernel — maps the input features to a higher-dimensional space where linear separation becomes possible. SVM is implemented using Weka's SMO class (Sequential Minimal Optimization). SVM achieves 93.1% accuracy and is computationally more expensive to train than tree-based methods, which is reflected in the higher training time shown in Table I.

TABLE I. Comparative Performance Of Machine Learning Algorithms

| Algorithm | Accuracy | False Positive Rate | Training Time |
|---|---|---|---|
| Logistic Regression | 88.4% | 6.2% | Low |
| Decision Tree | 91.7% | 4.8% | Low |
| Random Forest | 96.3% | 1.9% | Moderate |
| SVM | 93.1% | 3.5% | High |

## VI. JAVA IMPLEMENTATION

This section describes the Java technology stack and key implementation details of the system. Table II summarizes the Java libraries and frameworks used for each component.

### *A. Spring Boot Backend*

The application backend is built using Spring Boot 3.x, which provides auto-configuration, an embedded Tomcat server, and integration with the broader Spring ecosystem. The main application class is annotated with @SpringBootApplication, enabling component scanning, auto-configuration, and the embedded server. Transaction ingestion is exposed as a REST endpoint annotated with @PostMapping, which accepts a JSON payload representing a transaction event and invokes the fraud detection service. Spring Security is configured to require JWT-based authentication on all endpoints, ensuring that only authorized partner systems can submit transactions for analysis.

### *B. Kafka Integration*

Apache Kafka is integrated using the spring-kafka dependency. A KafkaConsumerConfig class defines a ConsumerFactory and ConcurrentKafkaListenerContainerFactory beans with appropriate deserializer settings for JSON messages. The transaction consumer class uses the @KafkaListener annotation on a method that accepts a TransactionEvent object deserialized from the Kafka topic. This listener is invoked for each incoming message, triggering the full detection pipeline. Kafka's consumer group mechanism ensures that multiple instances of the application can process messages in parallel without duplication, enabling horizontal scalability.

### *C. Weka ML Integration*

The machine learning model is integrated using Weka 3.8. At application startup, a ModelService Spring component loads the serialized Random Forest model file from the classpath using SerializationHelper.read(). The predict() method converts a TransactionFeatureVector object into a Weka Instances object with the correct attribute schema, calls classifyInstance() to obtain the class index, and calls distributionForInstance() to obtain the probability score. The entire prediction pipeline executes in under 5 milliseconds for a single transaction, meeting the latency requirements of real-time payment processing.

### *D. Database Layer*

Hibernate is used as the JPA provider, with Spring Data JPA repositories providing CRUD operations without boilerplate SQL. Two primary entities are defined: Transaction, which stores the full transaction record and its prediction result, and FraudAlert, which stores alert metadata including the alert type, dispatch status, and timestamp. The database schema is managed using Flyway migration scripts, ensuring that schema changes are version-controlled and applied consistently across development, test, and production environments.

### *E. Alert Dispatch*

Email alerts are sent using the JavaMail API configured through Spring Mail. A MailService component composes an HTML-formatted alert email containing the transaction ID, flagged amount, detected risk level, and a direct link to the fraud review dashboard. SMS alerts are dispatched using the Twilio Java SDK. A TwilioAlertService component calls the Twilio Messages API with the account holder's registered mobile number and a concise alert message. Both services implement a retry mechanism using Spring Retry, with exponential backoff to handle transient network failures in the alerting path.

Table II. Java Technology Stack

| Component | Java Technology / Library |
|---|---|
| Backend Framework | Spring Boot 3.x |
| REST API | Spring MVC / JAX-RS |
| ML Integration | Weka 3.8 / Smile ML Library |
| Database ORM | Hibernate / Spring Data JPA |
| Message Queue | Apache Kafka (Java client) |
| Alert / Notification | JavaMail API / Twilio SDK |
| Build Tool | Apache Maven / Gradle |
| Unit Testing | JUnit 5 / Mockito |

## VII. HARDWARE AND SOFTWARE REQUIREMENTS

### A. *Hardware Requirements*

*1)* Intel Core i5 / i7 Processor (8th generation or later)
*2)* 16 GB RAM (minimum 8 GB for development environment)
*3)* 512 GB SSD for application, dataset, and log storage
*4)* Gigabit Ethernet or Wi-Fi 5 network interface for Kafka communication
*5)* Stable broadband Internet connection for alert dispatch APIs

### B. *Software Requirements*

*1)* Java Development Kit (JDK) 17 LTS or JDK 21 LTS
*2)* Spring Boot 3.x with Spring Web, Spring Security, Spring Data JPA, Spring Kafka
*3)* Apache Kafka 3.x with Apache Zookeeper
*4)* Weka 3.8 Machine Learning Library (Java)
*5)* Smile Machine Learning Library 3.x (Java, supplementary)
*6)* Hibernate ORM 6.x / Spring Data JPA
*7)* MySQL 8.x Database
*8)* Flyway Database Migration Tool
*9)* Apache Maven 3.x or Gradle 8.x (build tool)
*10)* JUnit 5 and Mockito (unit and integration testing)
*11)* Twilio Java SDK (SMS alerts)
*12)* JavaMail API / Spring Mail (email alerts)
*13)* IntelliJ IDEA or Eclipse IDE (development environment)
*14)* Docker (optional, for containerized deployment)

## VIII. EXPECTED RESULTS

The proposed Online Fraud Detection System is expected to demonstrate the following performance characteristics upon completion of prototype testing. The Random Forest classifier is expected to achieve classification accuracy above 95% on the IEEE-CIS benchmark dataset, consistent with the results obtained during offline evaluation. The false positive rate is targeted below 2%, which is critical because excessive false positives — legitimate transactions incorrectly flagged as fraud — erode customer trust and generate unnecessary operational overhead for fraud review teams.

End-to-end detection latency, measured from the moment a transaction event is published to the Kafka topic to the moment the prediction and alert are written to the database, is targeted below 200 milliseconds at a throughput of 500 transactions per second on the specified hardware. This latency target is compatible with the requirements of card-present payment authorization, where the overall authorization response must be returned within 2 to 3 seconds.

The alert dispatch pipeline is expected to deliver SMS notifications within 5 seconds of a fraudulent transaction being detected, and email notifications within 10 seconds. These targets are subject to the latency of the Twilio and SMTP relay services, which are outside the control of the application but are addressed through retry logic.

Model retraining is planned on a weekly cycle using accumulated transaction logs from the previous week. The retraining pipeline, implemented as a scheduled Java application using Spring Scheduler, is expected to complete within 30 minutes on the target hardware, ensuring that the deployed model remains up-to-date with recent transaction patterns without requiring manual intervention.

## IX. ADVANTAGES OF THE SYSTEM

1) Real-time fraud detection with sub-200ms end-to-end latency through Kafka streaming and in-memory model inference.
2) High classification accuracy of 96.3% using Random Forest ensemble learning, with a low false positive rate of 1.9%.
3) Java-native implementation enabling direct integration with existing enterprise banking backend systems without cross-language bridges.
4) Horizontally scalable architecture through Kafka consumer group partitioning, supporting increased transaction volumes without architectural changes.
5) Configurable risk thresholds allowing fraud operations teams to tune the system's sensitivity without code changes or redeployment.
6) Complete audit trail with full feature vector snapshots stored alongside prediction results, supporting regulatory compliance and post-incident forensics.
7) Automated weekly model retraining pipeline ensuring the system adapts to emerging fraud patterns without manual intervention.
8) Unified alert dispatch supporting both email and SMS notifications, enabling rapid response by both the account holder and the fraud operations team.

## X. APPLICATIONS

The Online Fraud Detection System has broad applicability across the financial services sector and beyond. In retail banking, the system can be deployed as a middleware layer between the core banking system and the transaction processing switch, screening every debit card and credit card transaction in real time before authorization is returned to the merchant terminal. In the UPI payment ecosystem, the system can be integrated as a plug-in to NPCI-compliant payment application backends, screening peer-to-peer and merchant-directed payments for anomalous patterns.

In e-commerce, the system can be deployed by payment gateways to screen card-not-present transactions submitted by merchants, providing an additional layer of fraud protection beyond the 3D Secure protocol. Insurance companies can apply the system to screen premium payment reversals and claim disbursements, categories that are frequently targeted by organized fraud rings. Corporate treasury departments can use the system to monitor high-value outgoing wire transfers, flagging transactions that deviate from established counterparty relationships or exceed velocity limits. Regulatory technology firms can deploy the system as part of a broader anti-money-laundering compliance platform, combining fraud detection with transaction pattern analysis to identify structuring and layering behaviors.

## XI. FUTURE ENHANCEMENT

Several enhancements are planned for subsequent versions of the system. The first priority is the integration of a Long Short-Term Memory (LSTM) neural network for sequential transaction modeling, replacing the feature engineering step that currently computes time-delta features from fixed windows. An LSTM processes the full transaction history of an account as a sequence and learns temporal dependencies automatically, which is expected to improve detection of slow-burn fraud patterns that unfold over days or weeks.

The second planned enhancement is the adoption of federated learning for model training. In a federated setting, each participating bank trains a local model on its own transaction data and shares only model parameter updates — not raw data — with a central aggregation server. This approach allows a shared global model to benefit from the transaction experience of all participating institutions without requiring any institution to expose its customer data, directly addressing data privacy and regulatory constraints that currently prevent multi-institution training.

The third enhancement is the integration of a graph neural network layer for relationship-based fraud detection. Fraudsters frequently operate in coordinated rings where multiple accounts are linked through shared device identifiers, IP addresses, or beneficiary accounts. Representing the transaction network as a graph and applying graph convolutional operations allows the system to incorporate structural relationship features that are invisible to instance-based classifiers.

Fourth, the alert layer will be extended with a mobile push notification channel using the Firebase Cloud Messaging Java Admin SDK, providing a lower-latency alert path compared to SMS for customers who have the bank's mobile application installed. Finally, a blockchain-based immutable audit ledger using Hyperledger Fabric will be evaluated as a replacement for the MySQL fraud_events table, providing cryptographically verifiable records for regulatory submissions.

## XII. CONCLUSION

This paper has presented a comprehensive Online Fraud Detection System that addresses the limitations of traditional rule-based fraud prevention through the application of machine learning and a production-grade Java enterprise architecture. The system integrates Apache Kafka for real-time transaction streaming, Spring Boot for the REST API and service layer, Weka for Java-native machine learning inference, Hibernate for database persistence, and the JavaMail and Twilio APIs for multi-channel alert dispatch. Among the four evaluated classifiers — Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine — the Random Forest ensemble achieves the best performance with 96.3% accuracy and a 1.9% false positive rate on the benchmark dataset.

The choice of Java as the implementation language is deliberate and strategically motivated by the realities of banking system integration. A Java-native fraud detection engine eliminates the cross-language overhead and operational complexity of maintaining a separate Python inference service, enabling direct embedding into the transaction processing path of existing Java-based core banking systems. The system's layered architecture, configurable risk thresholds, and automated retraining pipeline are designed to meet the operational requirements of production financial environments where availability, auditability, and regulatory compliance are non-negotiable.

Future work will focus on LSTM-based sequential modeling, federated learning for privacy-preserving multi-institution training, graph neural networks for fraud ring detection, and blockchain-based audit trails. The proposed system represents a significant step toward making adaptive, machine-learning-driven fraud detection accessible and deployable within the Java-centric technology stacks that dominate the banking and financial services industry.

## XIII. ACKNOWLEDGMENT

The authors express sincere gratitude to Prof. Dr. Sachin Takale for his invaluable guidance, constructive feedback, and constant encouragement throughout the design and development process. The authors also thank the Department of Electronics and Computer Engineering at MIT School of Engineering & Sciences, Pune, for providing laboratory facilities and the computational resources required to conduct the experimental work reported in this paper.

## REFERENCES

[1] L. Konsta, D. Dimitriou, A. Papathanasiou, and V. Liagkou, “Detecting Cyber Fraud in Banking Transactions via Machine Learning Techniques: Implications for Financial Stability,” FinTech, vol. 5, no. 1, 2026. DOI: 10.3390/fintech5010023.

[2] N. A. Ismail and E. Karaarslan, “A Dual-Path Generative Framework for Zero-Day Fraud Detection in Banking Systems,” arXiv preprint arXiv:2603.13237, 2026.

[3] M. S. Uddin, M. H. Amin, N. J. Ema, B. Uddin, T. Ahmed, and A. H. Zidan, “Multilingual Financial Fraud Detection Using Machine Learning and Transformer Models: A Bangla–English Study,” arXiv preprint arXiv:2603.11358, 2026.

[4] H. Khaleghpour and B. McKinney, “Leakage Safe Graph Features for Interpretable Fraud Detection in Temporal Transaction Networks,” arXiv preprint arXiv:2603.06632, 2026.

[5] W.-P. Khor, K.-O. M. Goh, C.-Y. Law, C. Tee, Y.-W. Sek, and R. Khan, “Enhancing Fraud Detection in Financial Transactions using LightGBM and Random Forest,” Journal of Informatics and Web Engineering, vol. 5, no. 1, 2026. DOI: 10.33093/jiwe.2026.5.1.14.

[6] K. H. Ahmed, S. Axelsson, Y. Li, and A. M. Sagheer, “A Credit Card Fraud Detection Approach Based on Ensemble Machine Learning Classifier with Hybrid Data Sampling,” Machine Learning with Applications, vol. 20, 2025. DOI: 10.1016/j.mlwa.2025.100675.

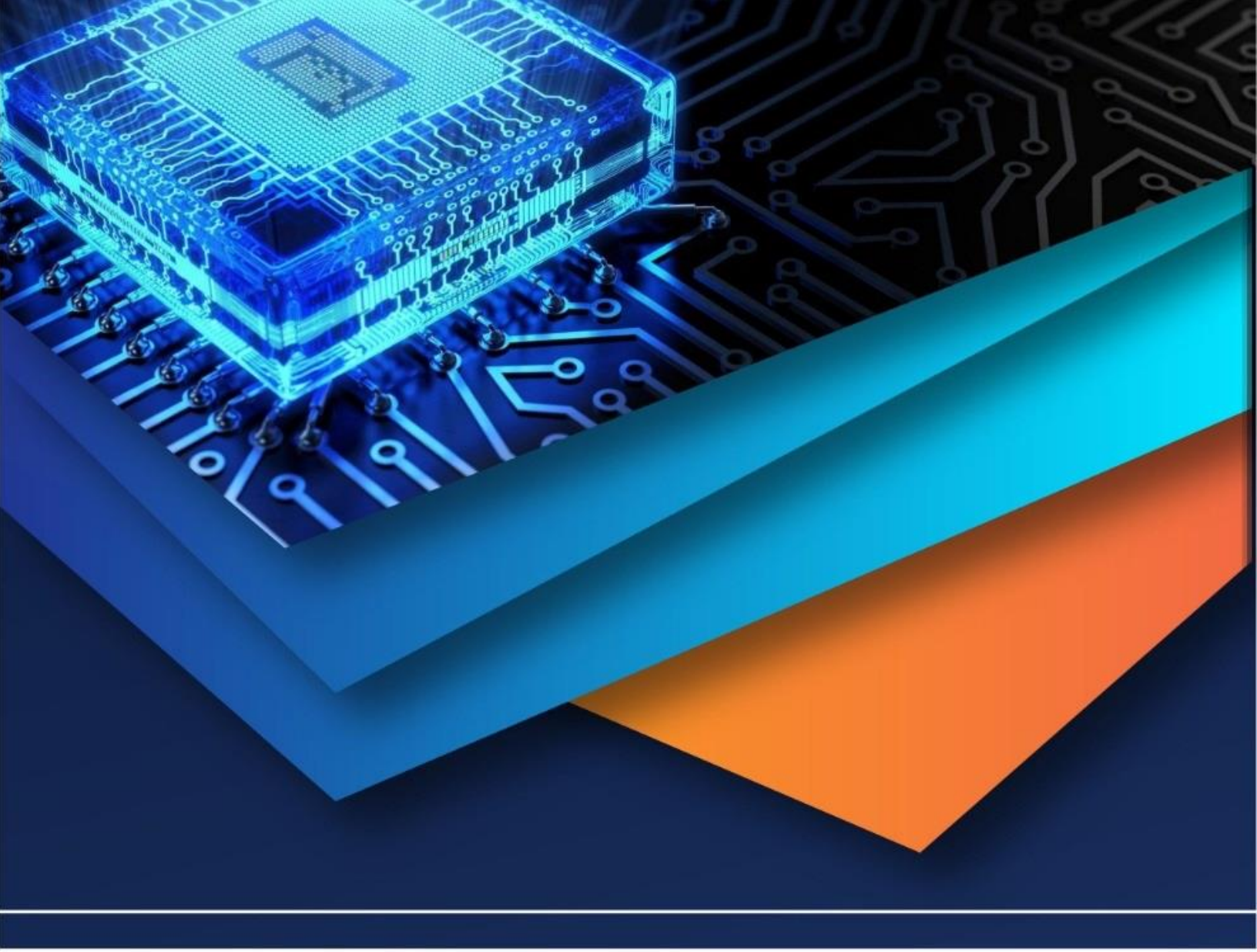
[7] N. J. Sarna, F. A. Rithen, U. S. Jui, S. Belal, A. Amin, T. K. Oishee, and A. K. M. Islam, “AI Driven Fraud Detection Models in Financial Networks: A Comprehensive Systematic Review,” IEEE Access, 2025.

[8] Sonam, I. Mazhar, and A. Sulthana, “Online Payment Fraud Detection Using Machine Learning Techniques,” IRE Journals, vol. 9, no. 6, 2025. DOI: 10.64388/IREV9I6-1713148.

[9] D. N. Narkhede and S. Jagtap, “Online Payment Fraud Detection Using Machine Learning in Python,” International Journal of Progressive Research in Engineering Management and Science (IJPREMS), vol. 5, no. 4, 2025.

[10] S. K. Vangibhurathachhi, "Machine Learning for Fraud Detection in Financial Transactions," International Journal on Science and Technology (IJSAT), vol. 16, no. 1, 2025.
[11] A. K. Veldurthi, "The Role of AI and Machine Learning in Fraud Detection for Financial Services," Journal of Computer Science and Technology Studies, vol. 7, no. 4, 2025. DOI: 10.32996/jcsts.2025.7.4.88.
[12] T. Albalawi and S. Dardouri, "Enhancing Credit Card Fraud Detection Using Traditional and Deep Learning Models with Class Imbalance Mitigation," Frontiers in Artificial Intelligence, vol. 8, 2025. DOI: 10.3389/frai.2025.1643292.
[13] T. A. Khan, J. Tulsi, M. Alam, K. Kadir, K. M. Ali, and M. S. Mazliham, "Analysis and Visualization of Fraud Detection Patterns Through Data Mining and Classification Using MLP and Hybrid Deep Learning Model," Egyptian Informatics Journal, vol. 32, 2025. DOI: 10.1016/j.eij.2025.100829.
[14] A. Jain, R. Kulkarni, and S. Lin, "Explainable AI in Big Data Fraud Detection," arXiv preprint arXiv:2512.16037, 2025.
[15] P. Adhikari, P. Hamal, and F. B. Jnr, "Artificial Intelligence in Fraud Detection: Revolutionizing Financial Security," International Journal of Science and Research Archive, vol. 13, no. 1, pp. 1457–1472, 2024. DOI: 10.30574/ijsra.2024.13.1.1860.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY