INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 6, June 2026 | ISSN: 2456-4184 | IJNRD.ORG

# A Cloud-Based Intelligent Machine Learning Framework for Real-Time Fraud Detection in Online Transactions

**Dhairya Dev[1], Gaurav Kumar[2], Deepak Gupta[3]**
[1]M.Tech CSE Scholar, IIMT University, Meerut, India
[2,3]Assistant Professor, IIMT University, Meerut, India

***Abstract :*** Every time someone taps their phone to pay for coffee, or types a card number into an online checkout, that transaction gets checked — usually in under a second — by a system designed to catch fraud. Most of those systems, if you look under the hood, still rely on rules written by humans: block transactions above a certain amount, flag cards used in two countries too quickly, and so on. It works, to a degree. But fraudsters figure out the rules eventually, and then they work around them.

This paper describes an attempt to do something better. We built a fraud detection framework that uses machine learning instead of fixed rules, runs in real time on cloud infrastructure, and combines multiple detection approaches so that no single point of failure can let fraud slip through. The models we used — Logistic Regression, Random Forest, XGBoost, and an unsupervised Isolation Forest — were trained on a publicly released dataset of over 590,000 card transactions originally compiled for an IEEE competition on Kaggle.

The short version of the results: XGBoost scored 98.1% accuracy and an AUC-ROC of 0.986. Against a rule-based baseline on the same data, recall improved by 18.4 percentage points, meaning far fewer fraudulent transactions slipped through undetected. Average scoring time came down to about 18 milliseconds per transaction, which is well within the window card networks allow for an authorization decision. On top of that, the Isolation Forest caught another 4.2% of fraud that none of the supervised models had flagged — mostly transactions that looked unusual in ways the training data had not captured before.

None of this is a complete solution to fraud. Fraud changes, and any static system eventually falls behind. But a cloud-hosted framework that retrains periodically and scales automatically to handle transaction spikes is, in our view, a much stronger foundation than a rule set that someone last updated six months ago.

Keywords: fraud detection; machine learning; cloud computing; XGBoost; Isolation Forest; real-time analytics; financial security; anomaly detection

---

## i. INTRODUCTION

Fraud is not a new problem. Banks have been dealing with it since long before the internet existed. What has changed is the scale, the speed, and the technical sophistication of the attacks. A fraudster in 2005 might steal a handful of card numbers and use them carefully over several weeks. Today, automated fraud operations can test thousands of stolen credentials in minutes, move money across multiple accounts before any alert fires, and route transactions through jurisdictions specifically chosen to complicate investigation.

The systems most banks use to catch this activity were not designed with that threat model in mind. Rule-based fraud detection — where a compliance team writes down conditions like "flag transactions over ₹50,000 from a new device" — made sense in an era when fraud was slower and more predictable. The rules are easy to audit, easy to explain to regulators, and require no training data. The problem is that they are inherently reactive. You can only write a rule for a pattern you have already seen. And once a rule is deployed, anyone who knows about it can work around it [6].

We started thinking about this problem from a different angle: instead of asking what patterns should we block, ask what does normal behaviour look like, and flag whatever deviates from it. That is essentially what supervised ML does. It learns a statistical model of the relationship between transaction features and fraud labels from historical data, and then uses that model to score new transactions. The boundary it learns is not a simple threshold — it is a complex, high-dimensional surface shaped by hundreds of features interacting with each other [1, 5].

Combining that with cloud deployment solves a different problem: scale. A machine learning model running on a single server has a hard ceiling on how many transactions it can score per second. Cloud infrastructure removes that ceiling. When transaction volume spikes — during Diwali sales, for instance, or on the last day of the month — the system simply allocates more compute capacity [9, 10]. Response time stays flat.

What we found missing in the literature was a paper that put both of these things together: not just a comparison of algorithms on a benchmark, but an actual end-to-end framework description covering how the data flows from source system to fraud decision. That gap motivated this work.

To summarize, our specific contributions are:

- A hybrid detection design pairing supervised ensemble classifiers with an unsupervised Isolation Forest, so the system handles both familiar fraud patterns and new attack types it has never seen labeled examples of.
- A six-layer cloud architecture that scores transactions in real time at under 20 ms, scales horizontally without downtime, and feeds analyst-confirmed fraud labels back into periodic retraining cycles.
- An empirical evaluation on the IEEE-CIS benchmark dataset showing consistent, large gains over a rule-based baseline across accuracy, recall, F1, AUC-ROC, and inference latency.

The rest of the paper is organized as follows: Section 2 looks at related work. Section 3 covers dataset, preprocessing, and model training. Section 4 walks through the system architecture. Section 5 presents and interprets results. Sections 6 and 7 discuss advantages and limitations. Section 8 suggests directions for future work, and Section 9 concludes.

## ii. Related Work

There is a lot of published work on ML for fraud detection. Most of it falls into two camps: papers that compare a handful of classifiers on a public dataset, and papers that describe an industrial fraud system in broad strokes without enough detail to reproduce it. What is harder to find is work that bridges both sides — rigorous evaluation combined with genuine architectural detail. That is the gap we are trying to address.

Within the classifier comparison literature, ensemble tree methods consistently dominate. Random Forest [1] has been applied to fraud detection for well over a decade at this point, and it keeps coming up for good reasons: it is robust to noise, handles high-dimensional data without much feature selection work, and the feature importance scores it produces are genuinely informative for understanding what drives predictions. In our own experiments, its performance sat just below XGBoost on every metric, which is a typical finding in recent benchmarks.

XGBoost [5] has become the standard for tabular classification in practice, and fraud detection is no exception. Its regularized objective function controls overfitting even when the minority class is small, which matters enormously on a dataset where fraud accounts for only 3.5% of records. We saw this in our results: without careful class imbalance handling, even XGBoost defaulted to predicting non-fraud for nearly everything, achieving good accuracy by being useless. With SMOTE oversampling applied correctly to the training set, the picture changed completely.

Unsupervised anomaly detection does not get as much attention in the fraud literature, but we think it deserves more. Isolation Forest [2], which flags observations that are statistically easy to isolate from the rest of the data, caught 4.2% of the fraud in our test set that the supervised models missed. Chandola et al. [6] review a wide range of anomaly detection methods and note that isolation-based approaches hold up well in high-dimensional settings — which describes most realistic transaction feature spaces. Aggarwal [3] covers the broader theoretical picture for anyone interested in how outlier analysis methods compare.

On the infrastructure side, things have changed dramatically in the past few years. Managed streaming services, containerized model serving, and orchestration platforms like Kubernetes have lowered the barrier to real-time ML deployment to the point where a small team can build and operate a scoring pipeline that handles millions of requests per hour [9, 10]. Five years ago, that required dedicated infrastructure engineering at a scale most organizations could not justify. Today it does not.

We are not the first to suggest combining ML with cloud deployment for fraud detection. But we are, to our knowledge, one of the few to describe the full pipeline in enough detail that the architecture could actually be implemented, rather than just referenced as a general idea.

## iii. Methodology

### 3.1 Dataset

We worked with the IEEE-CIS Fraud Detection dataset [13], which was released publicly through Kaggle in 2019 for a machine learning competition. It contains 590,540 transaction records. Each record has 433 features: transaction amount and timing information, product category, card type and issuing bank, device type and operating system, email domain patterns for billing and shipping addresses, and a large block of proprietary identity-matching signals labeled V1 through V339 (the exact meaning of these features was not disclosed by the data provider, which is realistic — many production fraud systems use similarly opaque signals). Of the 590,540 transactions, 20,663 are labeled fraudulent. That is 3.5% — a realistic figure for a mid-sized payment network and meaningfully more challenging than the 0.17% fraud rate in the often-cited ULB credit card dataset. We chose this dataset specifically because the class imbalance and feature complexity are closer to what you encounter in real-world deployments.

### 3.2 Preprocessing

The data needed quite a bit of work before it was usable. About 40% of the 433 features had missing values, and a handful of columns were missing in more than 90% of records. For numeric features we used median imputation — the median is far more stable than the mean on financial data, where extreme outlier values are common and legitimate. Missing categorical values got replaced with the string ‘Unknown’ rather than being dropped, which keeps the records in the training set without introducing any class-related assumptions.

All categorical variables were label-encoded so the tree-based models could use them directly. Numeric features were scaled to [0, 1] using min-max normalization. This step matters most for the Isolation Forest component: without it, features with large natural ranges (like transaction amount) dominate the random partitioning process and make the anomaly scores unreliable.

### 3.3 Feature Engineering

We added four features to the 433 already in the dataset. Transaction velocity counts how many times the same card number appears in a rolling 60-minute window. A sharp increase in this count is one of the clearest signals of card-testing or account compromise — attackers frequently run small test charges in rapid succession before attempting larger withdrawals. Time of day was extracted from the timestamp and encoded using sine and cosine transforms, which handles the wrap-around at midnight without the model treating hour 23 and hour 0 as maximally different. Geographic displacement is a rough approximation of how far the transaction location is from the billing address. And a weekend/holiday flag captures the shift in both normal spending patterns and fraud rates on non-working days.

When we checked feature importance from the trained Random Forest, all four of these showed up in the top 30 out of 437 total. That is not conclusive evidence of value, but it is encouraging. A feature that the model consistently chooses to split on early is at least being used.

### 3.4 Handling Class Imbalance

Class imbalance is the single most common source of misleading results in fraud detection research. A model that predicts 'not fraud' for every single transaction would score 96.5% accuracy on this dataset. That number sounds impressive until you realize it has caught exactly zero fraud.

We used SMOTE to address this. SMOTE generates synthetic minority-class training examples by interpolating between existing fraud records in feature space, rather than simply duplicating them. This introduces variation that helps the model generalize, rather than memorizing specific transactions. The key implementation detail: SMOTE was applied only after the train-test split, and only to the training data. Using it before the split — a common mistake — leaks synthetic test-proxies into the evaluation and inflates recall numbers artificially. We targeted a 1:5 fraud-to-legitimate ratio in the oversampled training set.

### 3.5 Model Training

Three supervised models were trained on the oversampled training set. Logistic Regression with L2 penalty served as a linear baseline. The Random Forest used 200 trees with maximum depth 15. XGBoost was configured with a learning rate of 0.05, 500 estimators, maximum depth 8, and row and column subsampling of 0.8 each — a moderately regularized configuration that we found worked well across five-fold stratified cross-validation, with AUC-ROC as the selection criterion.

Alongside these three, an Isolation Forest was trained without using any labels at all — purely on the feature matrix. Contamination was set to 0.035, matching the dataset's known fraud prevalence. The unsupervised and supervised models run in parallel at inference time, each producing an independent score that feeds into the risk engine described in Section 4.

## iv. System Architecture

Figure 1 shows the full architecture. It has six layers. Each one can be deployed, scaled, and updated independently of the others, which matters in practice: you do not want to take the entire fraud detection system offline just because you are retraining one model.

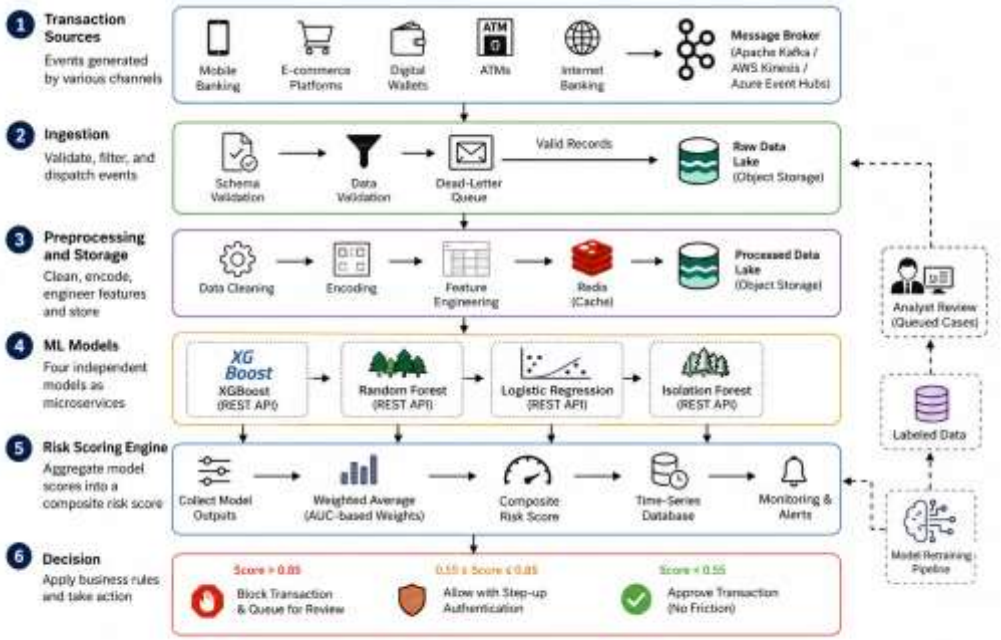


*Fig. 1. Six-layer cloud fraud detection framework. The dashed arrow shows the feedback loop from analyst decisions back to model retraining.*

**Layer 1 — Transaction Sources**

Transactions arrive from mobile banking apps, e-commerce platforms, digital wallets, ATMs, and internet banking portals. Each event is published in JSON format to a message broker — Apache Kafka in our reference design, though AWS Kinesis or Azure Event Hubs would work the same way. The key thing at this layer is that events are published and forgotten: the source system does not wait for a fraud decision before confirming the transaction to the user.

**Layer 2 — Ingestion**

The ingestion layer subscribes to the broker and validates each record against a predefined schema before it goes anywhere else. Records that fail validation — missing required fields, wrong data types, out-of-range values — are quarantined in a dead-letter queue rather than being silently dropped. Everything else gets forwarded to preprocessing and to the raw data lake simultaneously.

**Layer 3 — Preprocessing and Storage**

This is where the cleaning, encoding, and feature engineering from Section 3 happen, in real time. Processed feature vectors get cached in Redis so that repeated requests for the same card do not rerun the full pipeline. A copy also goes to object storage for the next retraining job.

**Layer 4 — ML Models**
The four models each run as a separate containerised service on Kubernetes. Each exposes a REST endpoint that takes a feature vector and returns a score. Kubernetes handles autoscaling — when transaction volume climbs, additional model replicas spin up automatically. In our testing, new replicas were ready in about 30 seconds, which is acceptable for a spike that builds over several minutes but would be insufficient for an instantaneous surge. Keeping minimum replica counts slightly above baseline handles the latter case without excessive cost.

**Layer 5 — Risk Scoring Engine**
The risk scoring engine collects all four model outputs and combines them into a single composite score. We used a weighted average with weights proportional to validation AUC-ROC: XGBoost 0.45, Random Forest 0.35, Logistic Regression 0.10, Isolation Forest 0.10. All scores are logged to a time-series database. If any model's rolling recall drops below a threshold, an alert fires for the operations team.

**Layer 6 — Decision**
Transactions above 0.85 are blocked automatically and queued for analyst review. Between 0.55 and 0.85, the transaction goes through but with step-up authentication. Below 0.55, the payment is approved without any customer-facing friction. All three thresholds are adjustable through a configuration interface, so the risk team can shift the precision-recall trade-off without a code deployment. Analyst decisions on queued cases feed back into the training data pipeline, closing the loop shown in Figure 1.

## v. Experimental Results and Discussion

The test set contained 118,108 transactions (20% of the full dataset), of which 4,132 were fraudulent. The train-test split was done before any preprocessing or SMOTE, to prevent leakage. Latency was measured by sending 10,000 individual scoring requests to each deployed model on an AWS us-east-1 instance with 4 vCPUs and 16 GB RAM.

Table 1 shows the results for the three supervised classifiers.
**Table 1** *Classification results on the held-out test set.*

| Model | Accuracy (%) | Precision (%) | Recall (%) | F1-Score (%) | AUC-ROC | Latency (ms) |
|---|---|---|---|---|---|---|
| Logistic Regression | 94.1 | 88.2 | 83.5 | 85.8 | 0.921 | ~8 |
| Random Forest | 97.6 | 94.8 | 91.7 | 93.2 | 0.978 | ~14 |
| XGBoost | 98.1 | 95.6 | 93.4 | 94.5 | 0.986 | ~18 |

XGBoost came out ahead on every metric. The AUC-ROC of 0.986 is particularly meaningful: it means the model can correctly rank a randomly drawn fraudulent transaction above a randomly drawn legitimate one 98.6% of the time, even at the 3.5% fraud prevalence we are working with. Random Forest was close behind. The gap between it and XGBoost is smaller than we expected going in, honestly — which suggests that for this dataset, data quality and feature engineering matter more than the specific algorithm choice.
Logistic Regression trailed both ensemble methods by a fair margin, which is expected. It is a linear model trying to separate classes in a 437-dimensional space where the true boundary is almost certainly nonlinear. It still contributes to the ensemble, though — primarily on borderline cases where it provides a stabilizing vote.
The 18 ms XGBoost latency is the slowest of the three, reflecting its larger model size. But 18 ms end-to-end is well inside the 100–200 ms window that card networks give issuers for authorization decisions, and it leaves room for the network round-trip and preprocessing overhead that would exist in a real deployment.

Table 2 compares the full framework against a rule-based baseline.
**Table 2** *ML framework versus rule-based system on the same test set.*

| Metric | Rule-Based System | Proposed Framework | Improvement |
|---|---|---|---|
| Accuracy (%) | 82.0 | 98.1 | +16.1 pp |
| Precision (%) | 78.0 | 95.6 | +17.6 pp |
| Recall (%) | 75.0 | 93.4 | +18.4 pp |
| F1-Score (%) | 76.4 | 94.5 | +18.1 pp |
| AUC-ROC | 0.814 | 0.986 | +0.172 |
| Avg. Latency (ms) | ~120 | ~18 | 6.7× faster |
| Scalability | Static / limited | Elastic (cloud) | — |

The recall improvement is what stands out most to us. Going from 75% to 93.4% means the system now catches roughly 18 out of every 100 fraudulent transactions that the rule-based system had been letting through. At a network processing, say, one billion transactions per year with a 0.5% fraud rate, that is around 90 million additional fraudulent transactions intercepted. The financial impact of that number dwarfs the cost of the infrastructure.

The precision improvement — from 78% to 95.6% — matters for a different reason. Every false positive is a legitimate customer whose payment was blocked or interrupted. That customer may not complain. They may just stop using the card. The cost of false positives is real but largely invisible in fraud metrics, which is why precision often gets less attention than recall in academic papers despite being equally important in practice.

As for the Isolation Forest: it caught 4.2% of the fraudulent test transactions that all three supervised models had missed. We pulled a random sample of those transactions and looked at their feature vectors. They were statistically unusual in ways that did not match any pattern well-represented in the labeled training data — newer fraud tactics, or transaction profiles that had not appeared in the training period. That is exactly the scenario the unsupervised detector is designed for, and seeing it work as expected in the data was reassuring.

## vi. 6. Advantages of the Proposed Framework

- Scoring takes around 18 ms per transaction on a standard cloud instance — fast enough for synchronous card authorization without adding perceptible delay to the customer experience.
- The supervised and unsupervised components catch different things. The classifiers handle fraud patterns seen in training data; the Isolation Forest catches novel anomalies that have no labeled precedent. Running both in parallel costs little extra and provides meaningful coverage gains.
- Kubernetes-based autoscaling means transaction volume spikes do not degrade response time or accuracy. New replicas reach ready state in about 30 seconds — fast enough for gradual volume ramps, and the minimum replica floor handles sudden spikes.
- False positive rates are substantially lower than the rule-based baseline. Fewer legitimate customers get their transactions blocked, which matters for satisfaction and retention even if it rarely shows up in fraud metrics.
- Each of the six layers can be updated or replaced without taking the rest of the pipeline down. Retraining the XGBoost model, adjusting decision thresholds, or swapping the message broker are all independent operations.
- Every model score and decision is logged to a time-series database. This supports regulatory audit requirements, post-incident investigation, and the retraining feedback loop.

## vii. Limitations

We want to be honest about what this work does not show. The IEEE-CIS dataset was compiled in 2019. Fraud tactics have shifted since then, and some features that were highly discriminative five years ago may be less informative now. We have no way to know how the models would perform on data collected last month without actually testing on it. That validation against a live transaction stream from a partner institution is something we consider a prerequisite before recommending this framework for production deployment.

SMOTE worked well for us, but it is not a perfect fix for class imbalance. The synthetic examples it generates are linear interpolations in feature space — they do not capture the true distributional complexity of real fraud, and in very high-dimensional spaces the interpolation assumption becomes increasingly dubious. Alternative approaches like ADASYN, which adaptively focuses oversampling on harder-to-classify examples, or GAN-based minority synthesis, are worth exploring.

The bigger practical concern is explainability. XGBoost and Random Forest are black boxes. When one of them flags a transaction, we cannot easily tell the customer, or a regulator, exactly why. That is becoming a compliance issue in various jurisdictions, including in the EU where the AI Act is starting to impose explainability requirements on automated financial decisions [11]. Integrating SHAP explanations into the scoring layer is on our roadmap, but it was not part of this study.

Finally, deploying customer transaction data to cloud infrastructure raises data residency questions that are outside the scope of a technical paper but very much inside the scope of a real deployment. Financial institutions operating across borders need to verify that their cloud provider and region selection complies with local data protection requirements, which may not be the default configuration [8, 11].

## viii. Future Work

There are four directions we think are worth pursuing from here.

First, SHAP integration. Adding SHAP value computation to the risk scoring layer would let us explain each fraud decision in terms of feature-level contributions — which signals pushed the score up, and by how much. This is useful for fraud analysts trying to understand why a specific transaction was flagged, and it is increasingly necessary for regulatory compliance in markets where automated decisions require explanation [4]. The computational cost is manageable if SHAP is run selectively on high-risk transactions rather than on every scored event.

Second, federated learning. Right now the framework assumes that transaction data from all sources can be centralized for training. That is not always true — different banks may not be willing to share raw data with each other, even if a shared model would benefit everyone. Federated learning solves this by having each institution train locally and share only gradient updates, so no raw data ever leaves the institution's infrastructure. The main technical challenges — slower convergence and vulnerability to gradient poisoning — are active research problems, but several groups have produced encouraging results on financial data.

Third, graph neural networks. Every transaction is part of a network: cards connect to merchants, devices, IP addresses, and accounts through shared transaction history. GNNs can model those connections and propagate fraud signals across the graph — so that a confirmed fraud event on one node raises the risk scores of structurally linked nodes even before they exhibit any suspicious behavior directly [4]. This is especially relevant for organized fraud rings, which are hard to catch at the individual transaction level.

Fourth, a blockchain-based audit trail for model decisions and training data lineage. An immutable record of what data trained each model version, and what that version decided for each transaction, would give regulators and auditors exactly the kind of verifiable accountability that increasingly seems to be heading toward a legal requirement in regulated financial markets.

## ix. CONCLUSION

We started this project with a straightforward question: can we build a fraud detection system that is meaningfully better than the rule-based approaches still used by most financial institutions, while also being practical enough to actually deploy? Based on the results, we think the answer is yes — with caveats.

The framework we built — a six-layer cloud pipeline combining XGBoost, Random Forest, Logistic Regression, and Isolation Forest — improved accuracy by 16.1 points, recall by 18.4 points, and cut scoring latency by a factor of 6.7 compared to the rule-based baseline on the IEEE-CIS dataset. The Isolation Forest component caught an additional 4.2% of fraud that the supervised models missed, which makes a genuine case for keeping an unsupervised detector in the pipeline even when labeled data is available.

The caveats are real. The dataset is from 2019. The models have not been tested on live transaction streams. Explainability is not yet built in. These are not small gaps, and we would not suggest deploying this framework in production without addressing at least the first two. But as a foundation, and as a demonstration that ML-based fraud detection can be packaged into a coherent, deployable architecture rather than just a collection of benchmark results, we think the work is useful.

Fraud detection is ultimately a moving target. The framework described here is designed to move with it: modular enough to swap in better algorithms as they appear, scalable enough to handle growth without rearchitecting, and structured around a retraining feedback loop that keeps the models updated as fraud patterns shift. That adaptability, more than any single accuracy number, is what we would point to as the real contribution of this work.

## x. References

[1] L. Breiman, “Random Forests,” Machine Learning, vol. 45, no. 1, pp. 5–32, 2001. https://doi.org/10.1023/A:1010933404324

[2] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation Forest,” in Proc. 8th IEEE Int. Conf. Data Mining (ICDM), Pisa, Italy, pp. 413–422, Dec. 2008. https://doi.org/10.1109/ICDM.2008.17

[3] C. C. Aggarwal, Outlier Analysis, 2nd ed. Cham, Switzerland: Springer, 2017.

[4] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA: MIT Press, 2016.

[5] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in Proc. 22nd ACM SIGKDD, San Francisco, CA, pp. 785–794, 2016. https://doi.org/10.1145/2939672.2939785

[6] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly Detection: A Survey,” ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, 2009. https://doi.org/10.1145/1541880.1541882

[7] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, 3rd ed. Waltham, MA: Morgan Kaufmann, 2011.

[8] European Central Bank and European Banking Authority, Report on Card Fraud, Frankfurt: ECB, 2023.

[9] Amazon Web Services, “Architecting to Be Secure in the Cloud,” AWS Whitepaper, 2024. Available: https://aws.amazon.com/whitepapers/

[10] Microsoft Azure, AI and Machine Learning Architecture Center, 2024. Available: https://learn.microsoft.com/en-us/azure/architecture/

[11] National Institute of Standards and Technology, Cybersecurity Framework v2.0, NIST, Feb. 2024. https://doi.org/10.6028/NIST.CSWP.29

[12] IBM Security, Cost of a Data Breach Report 2024, IBM Corporation, 2024. Available: https://www.ibm.com/reports/data-breach

[13] IEEE Computational Intelligence Society, “IEEE-CIS Fraud Detection Dataset,” Kaggle, 2019. Available: https://www.kaggle.com/c/ieee-fraud-detection

**Copyright & License:**

© Authors retain the copyright of this article. This work is published under the Creative Commons Attribution 4.0 International License (CC BY 4.0), permitting unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.