A Microservice-Based Architecture for Real-Time Credit Card Fraud Detection with Observability

Robson S. Santos[1], Robesvânia Araújo[1], Paulo A. L. Rego[1], José M. da S. M. Filho[1], Jarélio G. da S. Filho[2], José D. C. Neto[2], Nicksson C. A. de Freitas[2], Emanuel B. Rodrigues[1], Francisco Gomes[1] and Fernando Trinta[1]

[1]*Federal University of Ceará (UFC), Av. Humberto Monte, s/n, Pici – 60440-593 – Fortaleza – CE, Brazil*

[2]*Sidi – Institute of Innovation for Digital Society, Av. República do Líbano, 251 – 51110-160 – Recife – PE, Brazil*

fi

Keywords: Fraud Detection, Financial Systems, Microservice Architecture, Machine Learning, Observability, Scalability.

Abstract: The growth of real-time financial transactions has increased the demand for scalable and transparent fraud detection systems. This paper presents a microservice-based architecture designed to detect credit card fraud in real time, integrating machine learning models with observability tools to monitor operational behavior. Built on OpenTelemetry (OTel), the architecture enables detailed tracking of performance metrics, resource usage, and system bottlenecks. Experiments conducted in a cloud-based environment demonstrate the scalability and efficiency of the solution under different workloads. Among the tested models, XGBoost outperformed Random Forest in throughput and latency, handling over 25,000 concurrent requests with response times under 50 ms. Compared to previous work focused solely on model accuracy, this study advances toward real-world applicability by combining fraud detection with runtime observability and elastic deployment. The solution is open-source and reproducible, and it contributes to the development of robust data-driven systems in the financial domain.

1 INTRODUCTION

The rapid adoption of real-time payment systems has revolutionized financial transactions by increasing their speed, convenience, and accessibility. However, this transformation has also led to a dramatic rise in financial fraud, especially scams that exploit immediacy and user trust (Abdallah et al., 2016). In Brazil, one of the largest and fastest-growing real-time payment markets, fraud-related losses reached nearly USD 380 million in 2023 and are expected to increase by 40% by 2028 ACI Worldwide - Scamscope Report (2023). These challenges highlight the urgent need for fraud detection solutions that are accurate, fast, scalable, and resilient to real-world operational demands.

Fraud detection systems face multiple technical and operational challenges: handling large volumes of imbalanced and anonymized data, adapting to changing fraud patterns, minimizing false negatives, and ensuring the timely processing of high-throughput requests. Moreover, modern deployments are increasingly based on microservices, which introduce complexity in monitoring and diagnosing performance issues. Traditional monitoring tools offer limited visibility into distributed systems, where identifying latency spikes, resource bottlenecks, or degraded model performance is crucial.

This paper proposes a real-time fraud detection architecture based on microservices and enhanced by observability. Using OpenTelemetry (OTel) (Blanco, 2023), an open-source framework that provides standardized APIs, SDKs, and tools for collecting telemetry data, the system captures operational information across services and infrastructure layers, enabling detailed analysis of request paths, bottlenecks, and model behaviors. The architecture is modular, scalable, and designed to integrate different machine learning (ML) models for fraud prediction. We compare Random Forest and XGBoost under varying user loads and system configurations, using both classic ML metrics (accuracy, precision, F1 score, and recall) and telemetry-based indicators: CPU usage, response time and request throughput.

Our contributions are threefold:

- We present a microservice-based architecture for

Santos, R. S., Araújo, R., Rego, P. A. L., Filho, J. M. S. M., Filho, J. G. S., Neto, J. D. C., A. de Freitas, N. C., Rodrigues, E. B., Gomes, F., Trinta and F.
A Microservice-Based Architecture for Real-Time Credit Card Fraud Detection with Observability.
DOI: 10.5220/0013650000003967
In *Proceedings of the 14th International Conference on Data Science, Technology and Applications (DATA 2025)*, pages 748-755
ISBN: 978-989-758-758-0; ISSN: 2184-285X

real-time fraud detection with integrated observability.

- We compare two machine learning models under different data balancing techniques and load conditions, emphasizing recall and latency.
- We demonstrate how observability tools support model evaluation, bottleneck identification, and system scalability.

The remainder of this paper is organized as follows: Section 2 discusses background concepts on fraud detection and observability. Section 3 reviews related work. Section 4 introduces the real-time fraud detection architecture. Section 5 presents the conducted experiments. Section 6 details the machine learning pipeline. Section 7 reports model evaluation results. Section 8 describes the experimental setup. Section 9 compares our approach with state-of-the-art solutions. Section 10 concludes the paper and outlines future work.

2 BACKGROUND

2.1 Fraud Detection in Financial Systems

The rise of real-time payment systems has increased both the frequency and sophistication of financial fraud. Traditional rule-based systems are no longer sufficient to detect evolving fraud patterns, especially in scenarios where users, under social engineering pressure, authorize fraudulent transactions. Machine learning (ML) has emerged as a key technology to detect these fraudulent transactions by learning from historical data and identifying behavioral anomalies.

Despite their effectiveness, ML-based fraud detection systems face several challenges: highly imbalanced datasets that increase false negatives, evolving fraud patterns requiring continuous retraining, and anonymized features that limit model interpretability and feature engineering.

To address these issues, various techniques have been adopted, including random oversampling and undersampling, synthetic data generation such as SMOTE, feature selection based on importance scores, and ensemble methods. Among classification algorithms, tree-based models such as Random Forest (RF) and XGBoost have gained prominence due to their robustness, accuracy, and suitability for tabular and imbalanced data (Fatima et al., 2023).

2.2 Observability and Microservices in ML Deployments

With the growing deployment of fraud detection systems in cloud-native environments, microservice architectures have become the standard. Microservices promote scalability, modularity, and fault isolation by decoupling components such as transaction ingestion, enrichment, classification, and authorization into independent services. However, this architectural style also introduces complexity in terms of system monitoring, diagnosis, and performance tuning.

Traditional monitoring tools often fail to provide end-to-end visibility across services, especially when analyzing the behavior of deployed machine learning models under different workloads. This gap has led to the adoption of *observability*, a concept that goes beyond traditional monitoring by incorporating metrics, logs, and distributed traces to infer the internal state of systems from their external outputs.

OpenTelemetry (OTel) is currently the most widely adopted observability framework, providing language-agnostic SDKs and exporters for telemetry data. Combined with backends such as *Prometheus* for metrics and *Jaeger* for tracing, OTel enables the collection and correlation of performance data across microservices and infrastructure components. In ML deployments, observability supports runtime model evaluation, identification of bottlenecks, and validation of system behavior under high traffic or unexpected loads. This is especially critical in real-time fraud detection, where latency and scalability directly impact effectiveness.

3 RELATED WORK

Recent research on fraud detection has emphasized both model-level and architectural advancements. A critical area of focus has been handling class imbalance in fraud datasets. Mienye and Sun (Mienye and Sun, 2023) propose a deep ensemble framework combining GRU, LSTM, and MLP networks with synthetic oversampling to achieve high recall and adapt to evolving fraud patterns. Gupta et al.(Gupta et al., 2023) conduct a comprehensive comparison of balancing techniques, including SMOTE, showing improved generalization in classifiers such as XGBoost. Bakhtiari et al.(Bakhtiari et al., 2023) also explore gradient boosting models like LightGBM and LiteMORT, using metrics such as AUC-ROC, F1-score, and recall to evaluate fraud detection performance.

In terms of architectural design, several studies adopt high-throughput infrastructures for real-time fraud detection. The SCARFF framework (Carcillo et al., 2018) integrates Apache Spark, Kafka, and Cassandra to process streaming data at scale, enabling online detection over evolving transaction flows. Surianarayanan et al.(Surianarayanan et al., 2024) present a low-latency architecture using Kafka and Random Forest, emphasizing scalability and responsiveness. Thennakoon et al.(Thennakoon et al., 2019) explore pipelines with real-time user interfaces for fraud management under peak loads, while Menshchikov et al.(Menshchikov et al., 2022) examine architectural trade-offs in Big Data antifraud systems using Apache-based components. Prusti et al.(Prusti et al., 2021) propose a graph-based approach leveraging Neo4j to enhance prediction accuracy with relational patterns in transaction networks. Compared to these efforts, our architecture uniquely combines modular microservices, telemetry instrumentation, and dynamic scaling through Kubernetes.

Observability has gained relevance in ML system deployments. Karumuri et al.(Karumuri et al., 2021) outline challenges in large-scale traceability and propose structured practices for telemetry and bottleneck detection. Kosinska et al.(Kosińska et al., 2023) advocate for the convergence of observability and MLOps using tools like MLflow, Airflow, and OpenTelemetry. Majors (Majors et al., 2022) further supports the adoption of vendor-neutral observability frameworks to enable consistent visibility across services and infrastructure. Our solution incorporates OpenTelemetry for full-stack observability, although orchestration tools are not yet included, we consider their integration a direction for future work.

Some initiatives have explored architectural aspects more explicitly. Coelho et al. (Coelho et al., 2024) employ XGBoost to detect malicious activity in academic networks, but without a microservice-based deployment or integrated telemetry. Previous work by (Santos et al., 2023) proposed a multilayered architecture using Random Forest for fraud detection in a simulated bank environment, achieving high accuracy in controlled tests.

Compared to this previous work, the present study introduces several key advancements:

- **Model Evolution:** adopting *XGBoost* improves processing time and system responsiveness under high load.
- **Observability Integration:** the use of *OpenTelemetry*, combined with Prometheus and Jaeger, allows for real-time performance monitoring, traceability, and resource usage analysis.
- **Scalability Testing:** the system undergoes testing under multiple configurations of service replicas and concurrent user loads, enabling a realistic analysis of its elastic behavior.
- **Open-Source and Reproducibility:** all code, datasets, and deployment configurations are publicly available to support further experimentation and evaluation.

To the best of our knowledge, this is one of the first studies to integrate machine learning, microservices, and observability in a unified architecture tailored for real-time fraud detection in the financial domain.

4 PROPOSED ARCHITECTURE

The proposed system is based on a microservice-oriented architecture designed to enable real-time credit card fraud detection with high scalability, modularity, and observability. Each service is responsible for a specific task in the fraud detection pipeline, allowing for independent deployment, horizontal scaling, and fault isolation. Services communicate via REST APIs and are orchestrated using Kubernetes.

The architecture includes the following core services:

- **Transaction Ingestion Service:** receives transaction data from clients or upstream systems.
- **Feature Enrichment Service:** enhances incoming transactions with derived features and contextual metadata.
- **Fraud Detection Service:** applies the trained machine learning model to classify transactions as either fraudulent or legitimate.
- **Telemetry Service:** integrates OpenTelemetry agents to collect metrics, logs, and distributed traces.
- **Gateway/API Manager:** acts as the central entry point for all external requests.

Observability is achieved through **OpenTelemetry (OTel)**, in combination with **Prometheus** for metrics collection and **Jaeger** for distributed tracing. Together, these tools provide comprehensive, end-to-end visibility across all services, enabling fine-grained monitoring of request latency, model inference times, resource utilization, and inter-service dependencies.

5 MACHINE LEARNING PIPELINE

This section presents the proposed approach for detecting credit card fraud, detailing each stage of the machine learning pipeline. Figure 1 illustrates the workflow of the system. We describe the main components involved in data preprocessing, feature selection, and model development.

5.1 Dataset

This study uses the *IEEE-CIS Fraud Detection* dataset[1], designed for developing and testing fraud detection systems in e-commerce environments. The dataset comprises two main tables: one containing transaction details (e.g., amount and card data) and the other providing demographic and device-related information. Of the 434 available features, 400 are anonymized. Each transaction is labeled as either legitimate or fraudulent, enabling binary classification. Due to the size of the dataset, we restricted our analysis to the *train_identity* and *train_transaction* subsets, which were merged to form the working dataset.

5.2 Data Preparation

Due to the complexity of the dataset, which contains numerous columns, a significant amount of missing data, and alphanumeric variables, it was necessary to perform a data preprocessing step (Gupta et al., 2023). This process included several activities, such as: (i) removing columns with excessive missing data; (ii) imputing missing values using the median calculated for each attribute; (iii) removing duplicate rows; (iv) encoding categorical alphanumeric values into numerical values; and (v) grouping related attributes. After this stage, the dataset was reduced from 434 to 224 attributes.

5.3 Feature Selection

After data preparation, we applied a decision tree-based technique to determine feature importance, selecting attributes with a score above 0.01. This step reduced noise, improved model efficiency, and resulted in the 25 most relevant features.

Key transaction information (e.g., *transactionID*, *TransactionAmt*, *card1*, *addr1*, *P_emaildomain*) is submitted by the client during requests, while additional attributes (e.g., *C1*, *D1*, *M4*, *V317*) are retrieved from the database. These features include payment card details, counts, temporal differences, and anonymized behavioral patterns relevant for fraud detection.

[1]https://www.kaggle.com/competitions/ieee-fraud-detection

5.4 Pipeline Modeling

The labeled dataset was split into training (75%) and testing (25%) sets using *StratifiedShuffleSplit* to preserve class proportions. Four classification algorithms were evaluated: *Logistic Regression*, *Decision Tree (DT)*, *Random Forest (RF)*, and *XGBoost*, under three conditions: original imbalance, random undersampling, and random oversampling.

Hyperparameter tuning was performed using *grid search* with 5-fold cross-validation, where each model underwent 10 evaluation experiments. The best configuration based on accuracy was then used to retrain the model and evaluate it on the test set.

5.5 Best Model Selection

Once the experimental phase was completed, it became crucial to identify the most effective model based on specific performance metrics, which were calculated using the test set results. The evaluation metrics adopted include accuracy, precision, recall and F1-score.

Recognizing the importance of each metric, our evaluation emphasizes minimizing false negatives, given the critical nature of fraud detection. A false negative occurs when a fraudulent transaction is mistakenly validated as safe, posing a substantial risk. In contrast, false positives, though inconvenient, typically have minor consequences as they only temporarily block a legitimate transaction. Thus, recall and F1-score are prioritized in our selection process to ensure the integrity and reliability of the fraud detection system. All scripts and code developed during the experiments are publicly available on GitHub[2].

5.6 Model Evaluation

The best results were obtained using randomly oversampled data. Logistic Regression showed the lowest performance, while Decision Tree improved to 99.52% accuracy. Random Forest achieved 99.98% accuracy with near-perfect precision and recall, while XGBoost reached 99.84% accuracy and 100% recall, offering slightly lower accuracy but comparable overall effectiveness.

These results emphasize the importance of metrics like precision and F1-score in imbalanced datasets. XGBoost's consistent performance and faster inference justify its adoption in the proposed real-time fraud detection solution.

[2]https://github.com/robsonsants/Credit_Card_Fraud-Detection_Pipeline

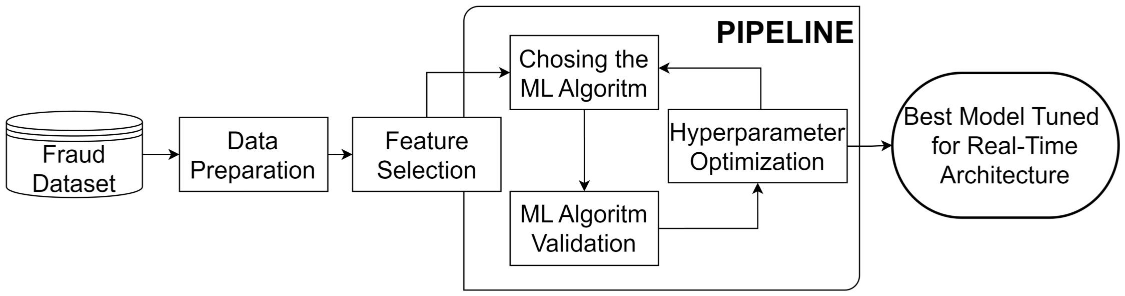

Figure 1: Machine learning model for fraud detection.

6 EXPERIMENTAL SETUP

The architecture was deployed on Amazon AWS using *t2.medium* and *c4.xlarge* EC2 instances. Services (`Manager`, `Data Enricher`, `Fraud Detection`, `Transaction Service`) were containerized with Docker and orchestrated via Kubernetes. Additional machines hosted the Locust load-testing tool.

The system was implemented in Python[3], employing libraries such as *FastAPI*, *Scikit-learn*, *NumPy*, and *PyMongo*. Docker and Kubernetes ensured consistent deployments and scalability.

Locust simulated 60 to 200 concurrent users. Observability was achieved with OpenTelemetry (OTel), Prometheus, and Jaeger, capturing latency, throughput, CPU usage, and traces.

As a limitation, experiments relied on a single dataset. Future work will explore multi-dataset evaluations and enhanced cross-validation.

7 REAL-TIME FRAUD DETECTION ARCHITECTURE

Figure 2 provides an overview of the proposed architecture, consisting of four microservices: `Manager`, `Data Enricher`, `Fraud Detection`, `Transaction Service`, and a `MongoDB` database.

The `Manager` service coordinates workflow orchestration (Megargel et al., 2021), initially receiving transaction data from clients. The `Data Enricher` adds additional features retrieved from MongoDB, aligning transactions with the attributes used for training.

The enriched transaction is classified by the `Fraud Detection` service using the XGBoost model. Based on the result, the `Manager` requests authorization from the `Transaction Service` and informs the client of the outcome.

Observability is integrated through OpenTelemetry (OTel). Services export telemetry data (metrics, traces, and logs) via an OTel *Collector*, enabling unified data processing and transmission based on the OpenTelemetry Protocol (OTLP).

Monitoring and visualization are supported by Prometheus[4] and Jaeger[5]. The system collects full-stack telemetry using three receivers (application, MongoDB, and host metrics) and two exporters (OTLP for Jaeger and Prometheus format).

7.1 Conducted Experiments

We evaluated system performance and scalability using Locust to simulate workloads and capture metrics such as response time, throughput, and CPU usage. Observability data collected via OTel, Prometheus, and Jaeger provided detailed logs of service behavior under load.

Experiments varied the number of clients (60, 100, 150, and 200), service replicas (1–2 for the manager and 1–3–4–5 for the transaction service), and machine learning models (Random Forest and XGBoost). Each run lasted 5 minutes, excluding the initial 10% of requests to mitigate cold start effects (Ebrahimi et al., 2024).

The next section presents the performance results under these conditions.

7.2 Experiment Results

The workload consisted of purchase transactions submitted by a variable number of clients. The evaluated metrics include average response time (ms), total requests processed, and peak CPU usage. These metrics allow us to assess the system's performance in terms of processing efficiency and scalability.

[3]https://github.com/robsonsants/Credit_Card_Fraud_Detection

[4]https://prometheus.io/

[5]https://www.jaegertracing.io/

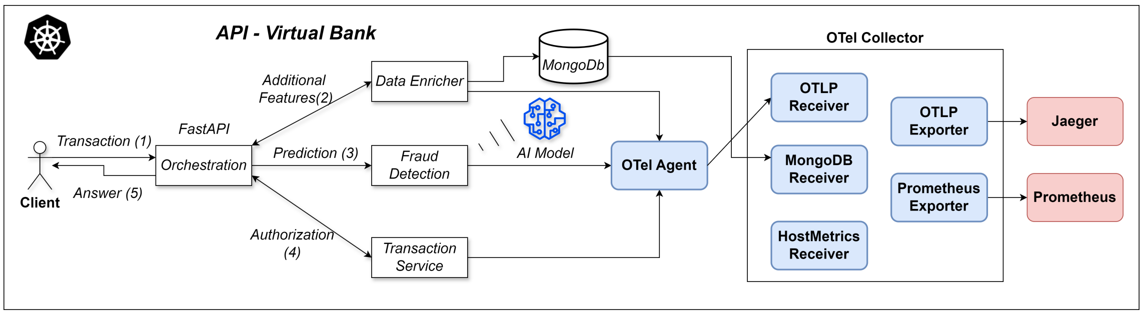

Figure 2: Real-time architecture for fraud detection.

Table 1: Performance metrics for Random Forest model.

Model Configuration	Users	Avg. Response Time (ms)	Total Requests	Max CPU (%)
RF_1_manager + 1_transaction	60	1454	4649	63
	100	3778	4712	64
	150	5655	4657	64
	200	6830	4694	64
RF_1_manager + 3_transaction	60	191	8148	110
	100	1181	9331	145
	150	2104	9345	145
	200	2747	9350	145
RF_2_manager + 4_transaction	60	137	8426	130
	100	735	11169	185
	150	1508	11205	185
	200	2039	11242	185
RF_2_manager + 5_transaction	60	121	8508	130
	100	714	11313	195
	150	1515	11151	195
	200	2029	11281	195

8 RESULTS AND DISCUSSION

8.1 ML Models Comparison

Table 2 and Table 1 compare model performance under varying system loads. Although Random Forest achieved the highest accuracy and F1-score, XGBoost demonstrated superior responsiveness and lower latency, with response times under 90 ms even for 200 users. Recall values above 99% for both models confirm their effectiveness, with XGBoost favored for production deployment due to its lighter footprint and consistent performance.

8.2 System Scalability

Scaling the number of replicas significantly impacted performance. As the number of `Transaction Service` replicas increased, the system handled more requests with lower latency. For example, deploying 5 replicas allowed XGBoost to process over 25,000 requests with sub-50 ms latency. This highlights the system's elastic scalability under Kubernetes, although dynamic auto-scaling policies remain a future improvement.

8.3 Observability Insights

Observability through OpenTelemetry (OTel) proved fundamental for diagnosing performance bottlenecks and understanding service behavior. Metrics and traces were collected from all components, including application services, MongoDB, and host infrastructure, and exported to Prometheus and Jaeger for analysis.

Jaeger traces revealed that the Random Forest model took approximately 71.41 ms per prediction, while XGBoost completed predictions in 26.77 ms, making it about 2.67 times faster. Prometheus metrics helped monitor CPU consumption and request throughput across different service configurations, supporting decisions about service scaling and resource allocation. Locust also contributed by generating workload and capturing per-request latency.

Table 2: Performance metrics for XGBoost model.

Model Configuration	Users	Avg. Response Time (ms)	Total Requests	Max CPU (%)
XG_1_manager + 1_transaction	60	30	9005	27
	100	35	16316	43
	150	47	21842	74
	200	82	25283	92
XG_1_manager + 3_transaction	60	31	8996	27
	100	37	16337	40
	150	49	21838	65
	200	90	25264	100
XG_2_manager + 4_transaction	60	28	9004	28
	100	33	16335	38
	150	38	21978	58
	200	46	25928	95
XG_2_manager + 5_transaction	60	29	9043	27
	100	33	16350	41
	150	39	21961	70
	200	48	25961	86

Overall, OTel-based observability enabled precise evaluation of system performance, guided the choice of the production model, and identified scalability opportunities. It played a key role in validating architectural decisions and preparing the system for real-world deployment. Additionally, distributed tracing allowed precise breakdown of service latencies, supporting real-time bottleneck identification.

9 COMPARISON WITH STATE OF THE ART

Our proposed architecture differentiates itself from existing fraud detection solutions by combining low-latency inference, integrated observability, modular microservices, and elastic scalability.

Unlike traditional ML orchestration tools such as **Airflow** and **MLflow**, which are oriented to batch processing and complex workflows, our approach targets real-time detection with immediate response, maintaining latencies below 50 ms.

Frameworks like **SCARFF** (Carcillo et al., 2018) and anomaly detection systems such as **RAVEN** (Coelho et al., 2024) achieve high accuracy but lack native observability and efficient scaling for real-time environments. Other studies (Menshchikov et al., 2022), (Thennakoon et al., 2019) explore architectural optimizations but do not integrate end-to-end monitoring or reproducibility.

The main differentiators of our architecture include:

- **Low-Latency Inference:** XGBoost-based model with sub-50 ms response under concurrent load.
- **Integrated Observability:** full-stack instrumentation with OpenTelemetry, Prometheus, and Jaeger.
- **Elastic Scalability:** dynamic service replication via Kubernetes.
- **Modular and Lightweight Design:** decoupled microservices with REST interfaces.
- **Reproducibility:** all code, configurations, and datasets are publicly available.

These attributes make our solution particularly suited for high-demand, latency-sensitive environments such as digital banking and real-time financial services.

10 CONCLUSION AND FUTURE WORK

This study proposed an observable and scalable architecture for real-time credit card fraud detection. Among the evaluated models, XGBoost outperformed Random Forest in response time, throughput, and CPU usage, confirming its suitability for high-demand environments.

The microservice-based design enabled modular deployment and horizontal scaling, while observability through OpenTelemetry facilitated real-time performance analysis and resource monitoring. Correlating telemetry with model performance provided a more holistic view for production deployments. The open-source availability of the solution further promotes reproducibility.

Future work will explore: (i) experiments on more powerful infrastructures; (ii) advanced balancing techniques like SMOTE; (iii) dynamic scaling and replication strategies; and (iv) broader evaluations using diverse datasets and comparisons with MLOps platforms such as MLFlow and Kubeflow.

ACKNOWLEDGEMENTS

This work was supported by the Brazilian Federal Agency for Support and Evaluation of Graduate Education (CAPES) – Finance Code 001.

The authors would like to thank FUNCAP – Ceará Foundation for Scientific and Technological Development Support – for the support provided throughout this work.

REFERENCES

Abdallah, A., Maarof, M. A., and Zainal, A. (2016). Fraud detection system: A survey. *Journal of Network and Computer Applications*, 68:90–113.

Bakhtiari, S., Nasiri, Z., and Vahidi, J. (2023). Credit card fraud detection using ensemble data mining methods. *Multimedia Tools and Applications*, pages 1–19.

Blanco, D. G. (2023). *Practical OpenTelemetry*. Springer.

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

Coelho, W., Zanotelli, V., Comarela, G., and Villaça, R. (2024). Raven: Detecção e classificação precoce de atores maliciosos em uma rede acadêmica. In *Anais do XLII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 351–364, Porto Alegre, RS, Brasil. SBC.

Ebrahimi, A., Ghobaei-Arani, M., and Saboohi, H. (2024). Cold start latency mitigation mechanisms in serverless computing: taxonomy, review, and future directions. *Journal of Systems Architecture*, page 103115.

Fatima, S., Hussain, A., Amir, S. B., Ahmed, S. H., Aslam, S. M. H., et al. (2023). Xgboost and random forest algorithms: an in depth analysis. *Pakistan Journal of Scientific Research*, 3(1):26–31.

Gupta, P., Varshney, A., Khan, M. R., Ahmed, R., Shuaib, M., and Alam, S. (2023). Unbalanced credit card fraud detection data: a machine learning-oriented comparative study of balancing techniques. *Procedia Computer Science*, 218:2575–2584.

Karumuri, S., Solleza, F., Zdonik, S., and Tatbul, N. (2021). Towards observability data management at scale. *ACM SIGMOD Record*, 49(4):18–23.

Kosińska, J., Baliś, B., Konieczny, M., Malawski, M., and Zielinśki, S. (2023). Towards the observability of cloud-native applications: The overview of the state-of-the-art. *IEEE Access*.

Majors, C., Fong-Jones, L., and Miranda, G. (2022). *Observability Engineering: Achieving Production Excellence*. O'Reilly Media, Incorporated.

Megargel, A., Poskitt, C. M., and Shankararaman, V. (2021). Microservices orchestration vs. choreography: A decision framework. In *2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 134–141. IEEE.

Menshchikov, A., Perfilev, V., Roenko, D., Zykin, M., and Fedosenko, M. (2022). Comparative analysis of machine learning methods application for financial fraud detection. In *2022 32nd Conference of Open Innovations Association (FRUCT)*, pages 178–186. IEEE.

Mienye, I. D. and Sun, Y. (2023). A deep learning ensemble with data resampling for credit card fraud detection. *IEEE Access*, 11:30628–30638.

Prusti, D., Das, D., and Rath, S. K. (2021). Credit card fraud detection technique by applying graph database model. *Arabian Journal for Science and Engineering*, 46(9):1–20.

Santos, R. S., Araújo, R., Rego, P. A., da SM Filho, J. M., da S Filho, J. G., Neto, J. D., de Freitas, N. C., and Rodrigues, E. B. (2023). Arquitetura de tempo real e modelo de aprendizado de máquina para detecçao de fraudes de cartao de crédito. In *Anais do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 265–278. SBC.

Surianarayanan, C., Kunasekaran, S., and Chelliah, P. R. (2024). A high-throughput architecture for anomaly detection in streaming data using machine learning algorithms. *International Journal of Information Technology*, 16(1):493–506.

Thennakoon, A., Bhagyani, C., Premadasa, S., Mihiranga, S., and Kuruwitaarachchi, N. (2019). Real-time credit card fraud detection using machine learning. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 488–493. IEEE.

APPENDIX

The metrics used and the results obtained are available at the following link[6].

[6]https://drive.google.com/file/d/1eAMEmJ2sQlHhzvDDOTZBvLujaWGRP0TD/view?usp=sharing