
This is the **published version** of the bachelor thesis:

Gratacós Paniello, Aleix. *A Full-Stack Machine Learning System for Fraud Detection in ERP Order-to-Cash (O2C) Processes*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334466>

under the terms of the  license.

A Full-Stack Machine Learning System for Fraud Detection in ERP Order-to-Cash (O2C) Processes

Aleix Gratacós Paniello

June 29, 2026

Resum– És difícil imaginar com funcionaria el món modern sense els sistemes *Enterprise Resource Planning* (ERP). Aquestes plataformes gestionen les operacions empresarials més importants, com ara la facturació, les vendes, els recursos humans i moltes altres. A causa de la dependència d'aquests programes i de l'auge de la intel·ligència artificial (IA), és de gran importància protegir aquests fluxos de treball contra activitats fraudulentament, que costen a les empreses d'arreu del món aproximadament el 5% dels seus ingressos anuals [1]. Més enllà dels enfocaments purament teòrics, aquest projecte té com a objectiu dissenyar i implementar una plataforma pràctica, *full-stack* i *end-to-end*, per a la detecció de frau i anomalies en els processos *Order-to-Cash* (O2C) d'un ERP. El sistema integra un generador de dades sintètiques, un injector de frau i un motor d'aprenentatge automàtic no supervisat basat en els algorismes *Isolation Forest* i *Local Outlier Factor*, combinat amb un *front-end* visual.

Paraules clau– ERP, Aprenentatge Automàtic No Supervisat, Detecció de Frau, Detecció d'Anomalies, *Order-to-Cash* (O2C), *Full-Stack*.

Abstract– It is difficult to imagine how the modern world would function without Enterprise Resource Planning (ERP) systems. These platforms manage the most important business operations, such as billing, sales, human resources, and much more. Due to the dependence on these programs and the rise of AI, it is highly important to secure these business flows against fraudulent activities, which cost companies worldwide approximately 5% of their yearly revenue [1]. Moving away from purely theoretical approaches, this project aims to design and implement a practical, end-to-end full-stack platform for fraud and anomaly detection within ERP Order-to-Cash (O2C) processes. The system integrates a synthetic data generator, a fraud injector, and an Unsupervised Machine Learning engine using Isolation Forest and Local Outlier Factor algorithms, coupled with a visual front-end.

Keywords– ERP, Unsupervised Machine Learning, Fraud Detection, Anomaly Detection, Order-to-Cash (O2C), Full-Stack.



1 INTRODUCTION AND OBJECTIVES

1.1 Introduction

THE Order-to-Cash (O2C) flow is one of the most critical components of any business. It begins when a customer places a sales order and concludes when the company receives the corresponding payment. It is typically the largest module within an Enterprise Resource Planning (ERP) system. Its scale makes it a highly targeted

attack vector, vulnerable to both external threats and insider fraud. In recent years, driven by the rise of AI, ERP systems have migrated from data centralization to data analysis. These Intelligent ERPs (iERPs) are highly capable of processing massive volumes of data using Machine Learning (ML) and other analytical tools to identify and mitigate fraud [2].

To solve this issue, this work presents an end-to-end system developed from the ground up, placing a strong emphasis on the architectural design of the Machine Learning solution. Furthermore, the development process describes the technical and strategic decisions an enterprise must consider when deploying an anomaly detection framework of this nature.

- E-mail de contacte: aleixgratacosp@gmail.com
- Menció: Tecnologies de la Informació
- Treball tutoritzat per: Marc Ortega Gil (Àrea de Ciències de la Computació i Intel·ligència Artificial)
- Curs 2025/26

1.2 Objectives

To achieve this solution, the project is structured around the following specific objectives:

1. Research and analyze five distinct real-world fraud scenarios within the O2C cycle to establish the baseline and scope of the system.
2. Develop a realistic ERP transactional log generator to simulate legitimate business operations.
3. Implement a fraud injection engine capable of integrating the previously defined patterns into the clean dataset.
4. Deploy an unsupervised machine learning pipeline using the Isolation Forest and Local Outlier Factor algorithms to evaluate and score transactional risk.
5. Build a web-based front-end interface to unify the previous steps, providing a visual and intuitive dashboard for security auditors to inspect and manage anomalies.

2 STATE OF THE ART

Legacy ERP architectures were originally designed as centralized relational databases with the primary goal of recording business transactions. Historically, these primitive systems operated at a low level, relying predominantly on manual data entry to maintain corporate logs. As computing became more accessible, these platforms began integrating more sophisticated modules, such as Manufacturing Resource Planning (MRP) and Supply Chain Management (SCM). This integration automated a lot of administrative workflows, which reduced operational costs and improved efficiency [3]. This change also provided better forecasting and decision-making capabilities for the directive roles. Nowadays, with the rise of Cloud Computing and Large Language Models, the market trends towards even greater automation and strategic insights.

Modern ERPs have increasingly included these analytical capabilities in their products to combat corporate financial crimes. For instance, SAP addresses these vulnerabilities through its *SAP Business Integrity Screening* module, and Microsoft has developed *Dynamics 365 Fraud Protection*. They both scan large amounts of data in real time for anomalies and compliance risks. To better understand how these platforms work, we need to first examine the main methods used to detect fraud in the industry.

2.1 Types of Fraud Detection

2.1.1 Rules-Based Systems

Traditionally, fraud detection frameworks have heavily relied on rule-based systems to safeguard transactional flows [2], [4]. These mechanisms operate on hardcoded conditional logic (if/else) designed to flag activities that violate predefined boundaries. While effective at catching certain errors, rule-based detection suffers from vulnerabilities when confronted with more sophisticated fraud. Because these engines rely on static thresholds, they are vulnerable to insider threats [5].

Rules-based systems require humans to manually write and update conditions. Because fraud tactics change daily, companies must constantly add new rules to the system. Over time, this creates a web of rules that can contradict each other.

This rigid approach creates two main problems. First, loose rules allow sophisticated malicious actors, who are aware of the rules in place, to enter. Second, strict rules can block legitimate customers, which causes false positives. When a system declines real customers, the company loses sales and customer trust. Additionally, when an order does not fit an existing rule, humans must review it manually. In summary, rules-based systems react to known threats but cannot adapt to new fraud tactics automatically, so they are not really scalable, and should not be used outside of specific instances. [6].

2.1.2 Supervised Machine Learning

This method trains algorithms using labeled data, where each input has a known, correct output. Its goal is to predict outcomes or classify new data based on learned patterns. It is mainly used for classification (placing data into categories) and regression (predicting numbers). It is called "supervised" because it requires human effort to label the data before training. While this preparation takes some time, these models are generally accurate. Common uses include spam detection and price prediction [7], [8].

Even though these types of algorithms are widely used for fraud detection with great success, their main problem is that in practice, they first need to get trained on a great amount of high-quality data. Also, because of their nature, they cannot detect new attack patterns if they have not been trained on them.

2.1.3 Unsupervised Anomaly Detection

This method works with unlabeled data, meaning the algorithm receives raw information without any provided answers. Its goal is to discover hidden patterns or structures on its own. It is mainly used for clustering similar items, finding associations, and simplifying complex data. Since the algorithm learns without a "teacher," human analysts must often review and validate the results. Although these models require significant computing power, they are effective for tasks like market segmentation and anomaly detection [7], [9].

2.2 Selected Unsupervised Algorithms

This project uses two specific unsupervised algorithms to detect fraud, Isolation Forest and Local Outlier Factor, the most used for anomaly detection. This section explains the basic concepts behind both methods.

2.2.1 Isolation Forest

This algorithm isolates anomalies instead of profiling normal data points. It builds random partition trees to split the dataset. Because anomalies have unusual values, they require fewer splits to separate from the rest of the data. As a result, they end up closer to the root of the tree. The model

uses this shorter path length to assign higher risk scores, making it very efficient at finding clear outlier [10].

2.2.2 Local Outlier Factor (LOF)

This algorithm measures how much the density of a data point deviates from its neighbors. It compares the local density of a transaction with the densities of the items around it. A transaction is flagged as a local outlier if its neighborhood is much less dense than others, this is useful for finding context-specific anomalies that look normal globally but unusual when compared to nearby data [10].

3 METHODOLOGY

3.1 Development Lifecycle

Due to the nature of the project, this work uses an iterative methodology divided into four main development phases. This approach allows for continuous feedback loops, in order to adjust parameters as needed.

3.2 Development phases

- **Phase 1:** Focused on defining the project scope, analyzing the five real-world fraud scenarios within the O2C flow, and designing the unified log structure.
- **Phase 2:** Focused on programming the baseline data generation engine (clean logs) and implementing the fraud injection system.
- **Phase 3:** Dedicated to deploying the unsupervised anomaly detection engine, optimizing the Isolation Forest and Local Outlier Factor algorithms, and adjusting their parameters.
- **Phase 4:** Focused on building the interactive web-based dashboard so that a potential auditor can easily view, inspect, and manage the detected alerts.

4 SYSTEM ARCHITECTURE

4.1 Unified O2C Log Design

To ensure the system's feasibility, instead of implementing a multi-table relational database system like most commercial ERPs, this project implements a unified transaction log architecture for the O2C cycle. The system operates at a high level of abstraction in which each log entry represents a unique sales order header rather than individual line items.

4.2 Modeled Fraud Scenarios

Based on this principle, five real-world fraud scenarios were defined and integrated into the system:

- **Ghost Sales and Channel Stuffing:** *Ghost sales* involve creating fictitious customer accounts and invoices to inflate performance indicators [11], [12]. Similarly, *channel stuffing* intentionally overships goods to inflate quarterly revenue before accounting closures [13], [14]. The detection engine targets these anomalies by analyzing high order volumes from

newly created `Customer_ID` inputs, checking for immediate status reversals via `Order_Status`, and detecting statistical billing spikes in `Total_Amount` in a short span.

- **Discount Manipulation:** *Discount Manipulation* happens when employees give unauthorized or unusually high discounts to certain customer accounts [15], [16]. The system detects this problem using two main approaches. First, it looks for basic outliers, which are discounts that are simply too high across the entire company. Then, it looks for anomalies by comparing `Discount_Percent` with `Product_Category`; for instance, a 20% discount might be completely normal for groceries but highly unusual for expensive electronics. Finally, the pipeline tracks suspicious connections between specific employees and clients by analyzing the combination of `User_ID` and `Customer_ID`.
- **Unusual Operational Hours:** Audit standards show that employees who commit fraud usually do it when there is less supervision to minimize detection risk [17]. To detect this, the system reads the `Order_Timestamp` field to look for suspicious activities. It specifically flags any sales orders that are created outside regular working hours, such as late at night, during weekends, or on official corporate holidays.
- **Transaction Splitting:** This fraud happens when someone breaks one large transaction into several smaller orders to avoid company approval rules [18], [19]. For example, if any sale over 1000€ needs a manager's manual approval, a user might split it into smaller pieces. The algorithm detects this by looking for several quick orders sent to the same `Customer_ID` where each `Total_Amount` value is just below that limit.
- **Premature Invoicing (Bill-and-Hold):** This practice, although legal in some cases, involves sending an invoice to a customer before the products are actually shipped, usually to make the company's short-term financial results look better than they actually are [20], [21]. Since a proper sale requires the goods to be sent out first, the system looks for logical errors in the timeline. Specifically, it flags any transaction where the `Invoice_Date` comes before the `Shipping_Date`.

Finally, based on the requirements previously mentioned, the structure of the logs for the project is the following:

TABLE 1: TRANSACTION LOG STRUCTURE

Field Name	Data Type
Transaction.ID	String
User.ID	String
Customer.ID	String
Product.Category	String
Total.Amount	Float
Discount.Percent	Float
Order.Timestamp	DateTime
Order.Status	Enum
Shipping.Date	Date
Invoice.Date	Date

4.3 Data Generation Module

The baseline dataset is generated using a custom Python class named `LogGenerator`. This module simulates legitimate business transactions over a one-year period (from January 1st to December 31st, 2025). The class exposes a single public method, `generate_clean_logs()`, which returns a populated Pandas `DataFrame`.

To construct realistic log entries, the class uses the `Faker` library to generate random customer names, alongside the native `datetime` module to handle timestamps. In general, the class works by generating random timestamps `n_logs` times strictly during normal business hours, sorting them chronologically in a list, and assigning a unique `Transaction.ID` to each record. Next, it populates identification variables like `User.ID`, `Product.Category`, and `Customer.ID` with completely random values to simulate typical daily company activity. In contrast, fields like `Total.Amount`, `Discount.Percent`, `Shipping.Date`, `Invoice.Date`, and `Order.Status` are calculated dynamically based on those previous values to maintain realistic business logic. Finally, an extra field named `is_fraud` is added at the end, which is always set to 0 within this class to create a clean baseline, help test the fraud injection functions, and serve as ground-truth labels for model training later.

4.4 Fraud Injection Module

Once the clean logs are generated, a secondary class named `FraudInjector` handles the data contamination process. This class accepts the Pandas `DataFrame` or a `.csv` file from the generator and exposes a single public method named `inject_all_frauds()`.

This public method coordinates the execution of six private functions, each corresponding to a defined fraud scenario: `_ghost_sales()`, `_channel_stuffing()`, `_discount_manipulation()`, `_unusual_hours()`, `_splitting()`, and `_premature_invoicing()`.

Each function receives a parameter specifying the exact number of anomalies to introduce into the clean dataset. During development, these functions were made to mix the fake data into the normal clean data instead of adding obvious values that stand out too much. This makes sure that single fake transactions are hard to see at first glance, mak-

ing the test more challenging for the machine learning models later.

5 MACHINE LEARNING PIPELINE AND AUDITOR INTERFACE

5.1 Feature Engineering

Before running the machine learning models, the raw transaction data must go through a feature engineering phase to convert the logs into numbers that the algorithms can process [22].

The machine learning engine is implemented using the `scikit-learn` library. To transform the raw O2C transaction logs into an readable format for unsupervised anomaly detection, a dedicated feature engineering phase is required.

Empirical testing demonstrated that using a single feature layout degrades performance because `Isolation Forest` and `Local Outlier Factor (LOF)` work on different mathematical principles. Therefore, two independent data preparation methods were developed within the `UnsupervisedFraudDetector` class.

5.1.1 Isolation Forest Feature Set

The `_prepare_data_iforest()` method creates a smaller dataset focused mainly on numbers and historical ratios. This reduction helps the model isolate outliers much more efficiently:

- `total_amount_log`: Applies a logarithmic formula calculated as $\log(1 + \text{Total Amount})$. This smooths out huge numbers and prevents massive sales figures from distorting the rest of the data distribution.
- `discount_percent`: The exact discount rate given to the order, used to find deviations.
- `discount_relative_impact`: Divides the current discount percentage by the average discount this user usually grants. A tiny value (0.01) is added to avoid mathematical errors. This variable directly highlights cases of discount manipulation.
- `amount_vs_history`: Compares the current purchase amount with the customer's historical average order value. This allows the model to spot sudden and unusual spikes in buying behavior.
- `days_for_shipping`: Calculates the number of days between the invoice date and the shipping date. This catches timeline mistakes, such as negative numbers caused by premature invoicing.

5.1.2 Local Outlier Factor (LOF) Feature Set

The `_prepare_data()` method creates the dataset for the LOF model. Note that this model also reuses the raw base variables from the previous set (`total_amount`, `discount_percent`, and `days_for_shipping`). To avoid repetition, the list below shows only the unique variables:

- `customer_avg_amount`: The average amount this customer usually spends. Together with the current order amount, it helps LOF see if a purchase is too big or unusual for them.
- `user_avg_discount`: The average discount this employee usually grants. It helps the model see if the current discount is too high for this specific worker.
- `order_hour`: The exact hour (0–23) of the transaction, used to detect actions taken outside normal working schedules.
- `is_december`: A simple flag (1 or 0) to show if the order happened in December, helping the model understand typical end-of-year changes.
- `seconds_since_last`: The time gap in seconds between consecutive orders from the same user to the same customer. This helps LOF spot quick repetitions, like split invoices.

Before training the models, both feature sets are normalized using the `StandardScaler` module. This sets the mean to 0 and the standard deviation to 1 for all variables, ensuring that massive numbers (like euros) and smaller ones (like hours or seconds) have the same weight and do not distort the calculations.

5.2 The Precision/Recall Tradeoff

In unsupervised anomaly detection, achieving a 100% detection rate (*High Recall*) is unrealistic. Attempting to capture every true positive increases the rate of innocent transactions flagged erroneously (*False Positives*). In a real corporate environment, an excess of false alarms induces *alert fatigue* and misallocates limited auditing resources.

Unsupervised algorithms can adjust detection sensitivity through model contamination parameters, but balancing this trade-off requires understanding the context of the solution.

For example, a medical system might prioritize minimizing false negatives (*High Recall*) [23], while a spam filter minimizes false positives (*High Precision*) [24]. For this enterprise auditing framework, maximizing precision is preferred to ensure that generated alerts represent high-probability risks, avoiding unnecessary investigation costs.

To evaluate the precision-recall trade-off empirically, the framework utilizes the F_β score, which calculates a weighted harmonic mean between precision and recall [25]:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} \quad (1)$$

The weighting parameter has been set to $\beta = 0.5$. This configuration weights precision twice as much as recall to adapt to the constraints of an auditing department, but in a real scenario, this must be determined by assessing the capabilities of the company.

5.3 Algorithm Analysis

The experimental evaluation used a dataset with a baseline fraud contamination of 2.35%. The development progressed through two major iterations.

In the first iteration, the models used a unified feature layout. The results showed that the Isolation Forest algorithm was inadequate, yielding a maximum precision of $\sim 20\%$ (and reaching 45% only when setting a very restrictive contamination threshold of 0.1%). On the other hand, the Local Outlier Factor (LOF) achieved a much better precision of $\sim 70\%$ under a 0.5% model contamination setting. The `_ghost_sales` scenario remained undetected by both models.

To diagnose this performance, the fraud injector tagged each anomaly with a distinct number from 1 to 6. This tracking made an interesting discovery. It revealed that the two algorithms complement each other: Isolation Forest detected a higher percentage of *Discount Manipulation* and *Premature Invoicing*, whereas LOF isolated *Unusual Hours*, *Channel Stuffing*, and *Transaction Splitting*.

Based on these results, we separated the features. Isolation Forest was configured with fewer features, focusing on numerical anomalies, while LOF kept all features. Under this different configuration, the individual models achieved the performance metrics detailed in Table 2.

TABLE 2: INDIVIDUAL MODEL PERFORMANCE SUMMARY

Algorithm	Ideal Contamination	Precision	Recall
Isolation Forest	0.93%	56%	22%
Local Outlier Factor	0.48%	69%	14%

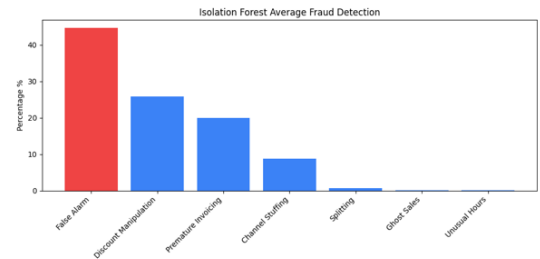


Fig. 1: Isolation Forest detection rates per fraud scenario. Bars in red represent False Alarms.

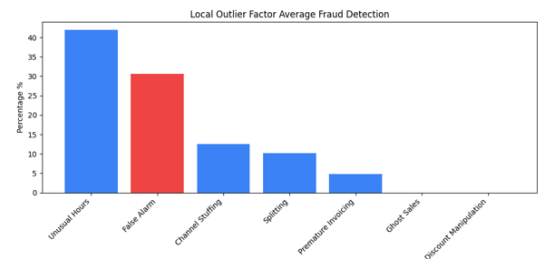


Fig. 2: Local Outlier Factor detection rates per fraud scenario. Bars in red represent False Alarms.

This specialization depends on the mechanics of each algorithm. Isolation Forest isolates numerical anomalies by constructing random decision trees; points that are far from the general distribution require fewer cuts

to be isolated. This makes it suitable for extreme values like high discounts or delivery dates. On the other hand, LOF measures metric distances against local neighbors (configured to $k = 20$). A sales order created at 06:00 AM is numerically close to one created at 09:00 AM, but LOF flags the transaction because it resides in a lower local density region compared to its closest neighbors.

Unfortunately, as can be seen in Figs. 1 and 2, the `ghost_sales` scenario remained undetected across both models due to its implementation logic.

In the generator, these transactions use standard amounts and categories, placing them in the middle of the global data distribution, which hides them from `Isolation Forest`. Also, since these bursts of transactions maintain temporal and numerical proportions identical to standard traffic, they preserve local density boundaries, eluding LOF. Adding features like customer account age or transactional velocity degraded the overall precision of the models and was discarded. In conclusion, as it is, stopping this type of fraud needs extra checks in the ERP system, such as tracking how warehouse stock moves or checking when payments are actually completed, which is out of scope for the project.

5.4 Ensemble Framework (Max-Aggregation)

Because each algorithm had its own strengths, an ensemble strategy was deployed to combine them both. The architecture implements a Max-Aggregation rule, which extracts the highest anomaly score from both methods and applies a maximum operator. The decision threshold that maximized the $F_{0.5}$ score was determined at the 99.5% percentile. The final ensemble metrics are summarized in Table 3.

TABLE 3: ENSEMBLE MODEL PERFORMANCE SUMMARY

Framework	Threshold	Precision	Recall
Ensemble	99.5%	67.7%	27.9%

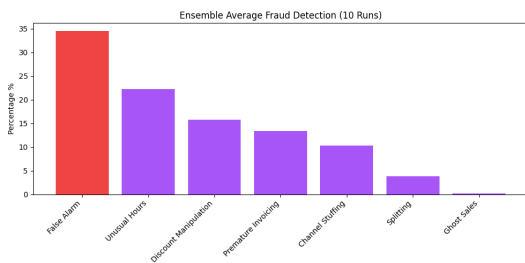


Fig. 3: Average fraud detection rates per scenario achieved by the Max-Aggregation ensemble framework across 10 evaluation runs. Bars in red represent False Alarms.

This implementation matched the precision of the LOF model while improving the recall capabilities.

5.5 Web Interface Architecture and Operational Walkthrough

To make it easy for an auditor to control and review the machine learning models, a web interface was developed. The frontend application is built as a Single Page Application (SPA) using the `Vue.js` framework and the `PrimeVue` ecosystem. The central application architecture uses `App.vue` as a global provider. It passes properties down to child views via `<router-view>` (such as `phases progress flags` and `detectionResults` objects) and listens to a custom `@changePhase` event to update the application state as the server responds.

The walkthrough of the user interface is structured around a step-by-step timeline, which is divided into six sequential steps:

1. Home Navigation (`HomeView.vue`): The flow is represented using a `Timeline` component. Each phase is wrapped inside a `Card` element. The interface evaluates if a step is open or closed using state checking methods. If a phase is completed, the timeline marker and the next action button is unlocked, to showcase the order of the simulation.

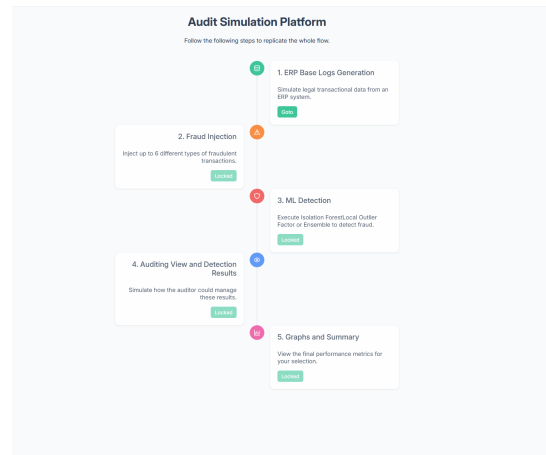


Fig. 4: Home view of the simulation platform showing the timeline.

2. ERP Base Logs Generation (`GeneratorView.vue`): This view enables the creation of standard business activity. The user introduces a target volume into an `InputText` box (which defaults to 100,000 records) and triggers a POST request to the server. The server responds with the amount of transactions created and saves the database to disk.
3. Fraud Contamination (`InjectorView.vue`): This page generates a grid layout form where the auditor specifies the exact number of anomalies to inject across the six defined fraud scenarios. Submitting the form updates the central state with the total count of contaminated rows.

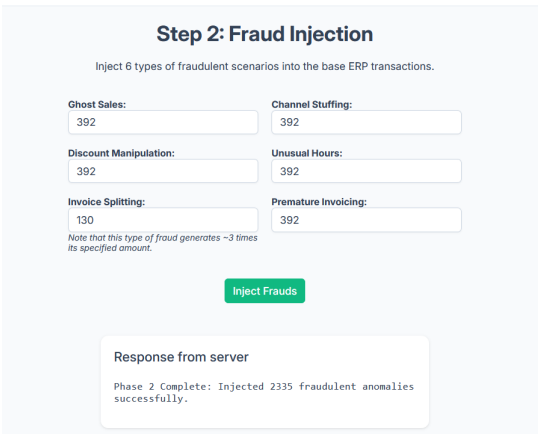


Fig. 5: View of the fraud injection step. For each type of fraud, the desired amount can be selected.

4. Machine Learning Fraud Detection (Detection-View.vue): This page lets the auditor choose and configure which machine learning models to run. The user can select the Ensemble Model, Isolation Forest, or Local Outlier Factor (LOF) using simple checkboxes, and manually type parameters like the threshold score or contamination rates into input fields. Clicking the execution button fires the requests to the Flask API.



Fig. 6: Machine Learning Fraud Detection interface showing the configuration options for ensemble and individual model parameters.

5. Results Dashboard and Forensic Auditing (AlertsView.vue): Once the analysis finishes, this page organizes all detected anomalies inside three separate tabs using paginated DataTable components. If the Ensemble model finds results, it automatically unlocks an option to open the Auditor Simulation Dashboard. This auditing view displays the main information cards summarizing the Total Alerts, Reviews Pending, and Total Amount at Risk, letting compliance officers review individual transaction tables and use interactive buttons to mark rows as either “Legitimate” or “Fraud” in real time.

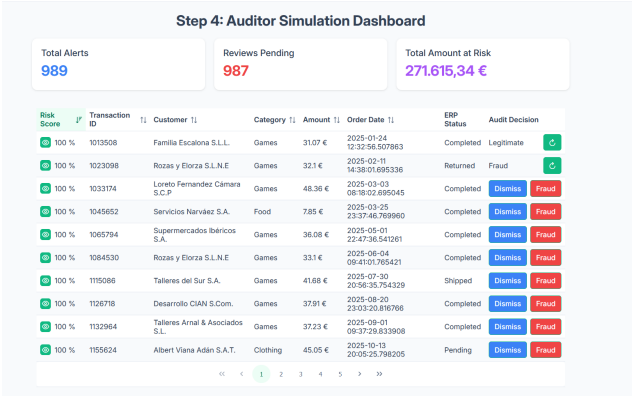


Fig. 7: Phase 4 Auditor Simulation Dashboard with summary cards.

Also there is a green button on the left that, when pressed, shows you all the details of the log, so you can see the transaction data in full (And not the summarized version).

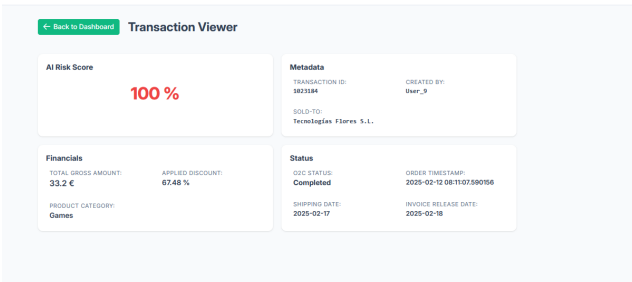


Fig. 8: Page that shows the full details of the selected log.

6. Performance Analytics and Matrices (GraphView.vue): The final analytics dashboard evaluates model execution against the ground-truth logs. The view requests the performance matrices from the backend.

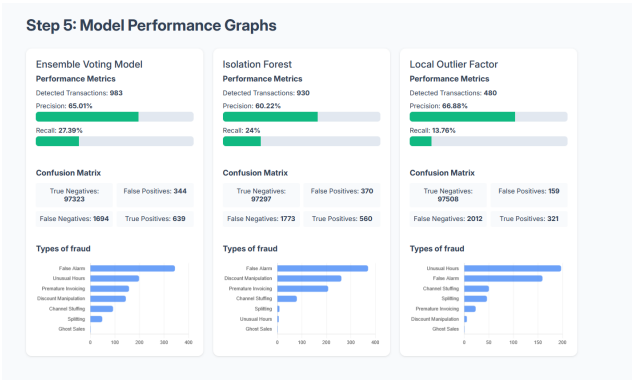


Fig. 9: View for seeing the performance of the models. For each model, precision, recall, the confusion matrix and the breakdown of the types of fraud detected can be seen.

6 CONCLUSIONS

This project successfully designed and implemented a complete full-stack auditing platform to detect fraud in ERP Order-to-Cash (O2C) processes. By creating a custom data

generator and a fraud injector, we established a simulation framework with a 2.35% fraud contamination rate to test unsupervised machine learning models.

The results proved that combining multiple algorithms is much more effective than relying on a single model. Isolation Forest excelled at identifying clear numerical outliers, such as unauthorized discounts and premature invoicing dates. On the other hand, the Local Outlier Factor (LOF) was ideal for detecting context-dependent anomalies, like unusual working hours, channel stuffing, and transaction splitting. By deploying a Max-Aggregation ensemble model with a 99.5% threshold, which is the threshold that maximizes out the F_β score, with $\beta = 0.5$, the system achieved a high precision of $\sim 67.7\%$ while significantly improving the recall rate to $\sim 27.9\%$.

However, the *Ghost Sales* scenario remained a blind spot for both models because these fraudulent orders perfectly matched normal data distributions and density boundaries. This limitation shows that stopping this specific type of fraud requires extra ERP controls outside the scope of this project, such as tracking physical stock movements or payment clearances.

Finally, the development of the web interface using `Vue.js` and `Flask` demonstrated how complex machine learning results can be turned into a practical tool. The framework and the auditor simulation dashboard allow compliance teams to easily review high-risk alerts and record real-time security decisions without dealing with the underlying complexity.

6.1 Future work

In a real world scenario, to improve on this project, a hybrid approach using both unsupervised and supervised Machine Learning could be used [26]. The logic would be the following:

First, a huge amount of data should be labeled. However, instead of wasting time labeling millions of normal transactions manually, the unsupervised engine developed in this work can act as an initial filter. The system flags the highest-risk anomalies, and the auditor uses the dashboard to review them, saving their decisions as either “Fraud” or “Legitimate”.

This data would create a labeled dataset over time. This labeled data can be used to train a supervised classification model. This second model can learn the exact patterns of confirmed fraud cases, drastically reducing false alarms and making the entire auditing architecture more efficient.

Finally, this supervised model could be used as an initial screening for very similar fraud cases and then the unsupervised model would focus on identifying unknown fraud patterns.

7 AI USE DISCLAIMER

Generative artificial intelligence was used solely to improve the formal aspects of this work, such as drafting, style, and spelling. Regarding technical aspects, it was utilized for specific assistance. Under no circumstances was it used for the essential intellectual and creative aspects of this project.

REFERENCES

- [1] Association of Certified Fraud Examiners (ACFE), “The nations @ occupational fraud 2024: A report to the nations,” ACFE, Tech. Rep., 2024.
- [2] S. Shiwakoti, “The role of intelligent erp systems in preventing corporate fraud and strengthening financial governance,” *Journal of Knowledge Learning and Science Technology*, vol. 4, pp. 138–147, Aug. 2025. DOI: 10.60087/jklst.vol4.n3.014
- [3] S. V. Mhaskey, “Integration of artificial intelligence (ai) in enterprise resource planning (erp) systems: Opportunities, challenges, and implications,” *International Journal of Computer Engineering in Research Trends*, vol. 11, pp. 1–9, Dec. 2024. DOI: 10.22362/ijcert/2024/v11/i12/v11i1201
- [4] A. P. Palanisamy, “Advanced rule-based fraud detection systems in payment processing,” *Journal of International Crisis and Risk Communication Research*, pp. 584–593, Oct. 2025, ISSN: 2576-0017. DOI: 10.63278/JICRCR.VI.3397
- [5] Fraud.net, *What is rules-based fraud detection? definition and guide*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.fraud.net/glossary/rules-based-fraud-detection#strengths-and-vulnerabilities>
- [6] C. Lovett, *Rules-based vs machine learning in fraud protection*, 2025. accessed Jun. 9, 2026. [Online]. Available: <https://www.signifyd.com/blog/rules-based-vs-machine-learning-fraud-protection/>
- [7] IBM Think Topics, *Supervised vs. unsupervised learning: What’s the difference? — ibm*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning>
- [8] Wikipedia Contributors, *Supervised learning - wikipedia*, 2026. accessed Jun. 9, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Supervised_learning
- [9] Wikipedia Contributors, *Unsupervised learning - wikipedia*, 2026. accessed Jun. 9, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Unsupervised_learning
- [10] S. Bhaumik, *Scalable anomaly detection: A deep dive into isolation forest vs local outlier factor — medium*, 2025. accessed Jun. 9, 2026. [Online]. Available: <https://medium.com/@sidbhaumik1918/scalable-anomaly-detection-a-deep-dive-into-isolation-forest-vs-local-outlier-factor-c1bac72dc133>

- [11] Oracle NetSuite, *Financial statement fraud: Understanding, detection and prevention — netsuite*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.netsuite.com/portal/resource/articles/accounting/financial-statement-fraud.shtml>
- [12] Billtrust, *Accounts receivable fraud: How to detect and prevent it — billtrust*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.billtrust.com/resources/blog/accounts-receivable-fraud>
- [13] Forensic Risk Alliance, *Revenue recognition frauds for lawyers: Channel stuffing*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.forensicrisk.com/news-and-insights/revenue-recognition-frauds-for-lawyers-channel-stuffing>
- [14] Forensic Risk Alliance, *Fraud prevention in revenue recognition*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.forensicrisk.com/news-and-insights/fraud-prevention-in-revenue-recognition>
- [15] P. Curwell, *Sales fraud: Protect your revenue and reputation*, 2025. accessed Jun. 9, 2026. [Online]. Available: <https://paulcurwell.com/2025/04/08/the-hidden-threat-to-your-bottom-line-how-sales-fraud-is-bleeding-your-business-dry/>
- [16] Chargeflow, *Return fraud in 2025: How retailers can fight back against \$103b in ecommerce scams*, 2025. accessed Jun. 9, 2026. [Online]. Available: <https://www.chargeflow.io/blog/return-fraud-exposed>
- [17] open-appsec, *Insider threat detection: The essential guide — open-appsec*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.openappsec.io/post/insider-threat-detection-the-essential-guide>
- [18] Houseblend, *Building custom rule engines for fraud detection in netsuite — houseblend*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.houseblend.io/articles/netsuite-fraud-detection-rules>
- [19] SafePaaS, *Prevent risk in procure to pay - safepaas*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.safepaas.com/articles/prevent-risk-in-procure-to-pay/>
- [20] Forensic Risk Alliance, *A revenue recognition classic – bill-and-hold arrangements*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.forensicrisk.com/news-and-insights/a-revenue-recognition-classic---bill-and-hold-arrangements>
- [21] Hubifi, *Auditing revenue recognition: A complete guide*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.hubifi.com/blog/how-to-audit-revenue-recognition>
- [22] IBM Think Topics, *What is feature engineering? — ibm*, 2026. accessed Jun. 9, 2026. [Online]. Available: <https://www.ibm.com/think/topics/feature-engineering>
- [23] A. Upadhyay, *Precision/recall tradeoff — analytics vidhya — medium*, 2022. accessed Jun. 9, 2026. [Online]. Available: <https://medium.com/analytics-vidhya/precision-recall-tradeoff-79e892d43134>
- [24] dataXmlchronicle, *Precision vs recall: Understanding the trade-off and why the f1 score matters — code like a girl*, 2021. accessed Jun. 9, 2026. [Online]. Available: <https://code.likeagirl.io/precision-vs-recall-trade-offs-importance-and-the-role-of-the-f1-score-a-must-know-interview-4a6516c26d52>
- [25] Wikipedia Contributors, *Precision and recall - wikipedia*, 2026. accessed Jun. 9, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Precision_and_recall
- [26] A. Jadhav, “Hybrid approaches in fraud detection: Combining supervised and unsupervised learning,” *International Journal of Scientific Research and Engineering Trends*, vol. 11, pp. 2395–566, 2022.