Combining Supervised and Unsupervised Learning for Efficient, Explainable, and Interoperable Anomaly Detection

Grisha Weintraub*, Leonid Rise, Doron Hillman and Paula Ta-Shma

IBM, Israel

Abstract

Anomaly detection is a widely used technique in various domains. While much of the research on anomaly detection focuses on improving detection accuracy and optimizing the training process, our approach emphasizes practical considerations, specifically addressing explainability, interoperability, and efficiency in real-world applications. Explainability refers to how clearly users can understand the reasoning behind why a specific point is labeled as an anomaly, interoperability pertains to how easily the anomaly detection module can be integrated into an application, and efficiency relates to the overhead associated with using the anomaly detection module in an application. We propose a practical approach that combines both supervised and unsupervised learning to enhance all three of these aspects. Our solution has been deployed in a real-world application for six months, and in this paper, we share the lessons learned during this period.

Keywords

Anomaly Detection, Machine Learning, Isolation Forest, Decision Tree

1. Introduction

Anomaly detection (AD) plays a vital role across various real-life applications. It is extensively used in the financial sector to identify fraudulent activity [1], in cybersecurity to detect potential threats [2], in medicine to recognize pathological processes [3], and in numerous other fields [4]. The high-level architecture of such applications looks as in Fig. 1:

1. A user submits a request to the application, which could be a credit card transaction or the results of a specific medical test.
2. The application receives the request, extracts critical data from it, and stores this information in a database. For instance, important data for a credit card transaction may include location, time, and amount. In the context of a medical test, relevant data could consist of various metrics like blood count.
3. A periodic offline process trains on the data in the database to build an anomaly detection model.
4. The model is then made accessible to the application.
5. The application utilizes the model to determine in real-time whether a particular user request is an anomaly.
6. Based on the model's response, the application makes a business decision, such as blocking a credit card transaction or sending an email to a patient or their doctor.

This architecture, while simple, has several practical limitations that lead to the following research questions addressed in this paper:

- `Q1 (Explainability):` The AD model operates as a black box, making it difficult for users to understand why a specific point was identified as an anomaly. Can we enhance its explainability?

Published in the Proceedings of the Workshops of the EDBT/ICDT 2026 Joint Conference (March 24-27, 2026), Tampere, Finland
*Corresponding author.
grisha.weintraub@ibm.com (G. Weintraub); leonid.rise@ibm.com (L. Rise); doron.hillman@ibm.com (D. Hillman); paula@il.ibm.com (P. Ta-Shma)
0000-0003-4823-4757 (G. Weintraub); 0009-0005-0298-6621 (L. Rise)

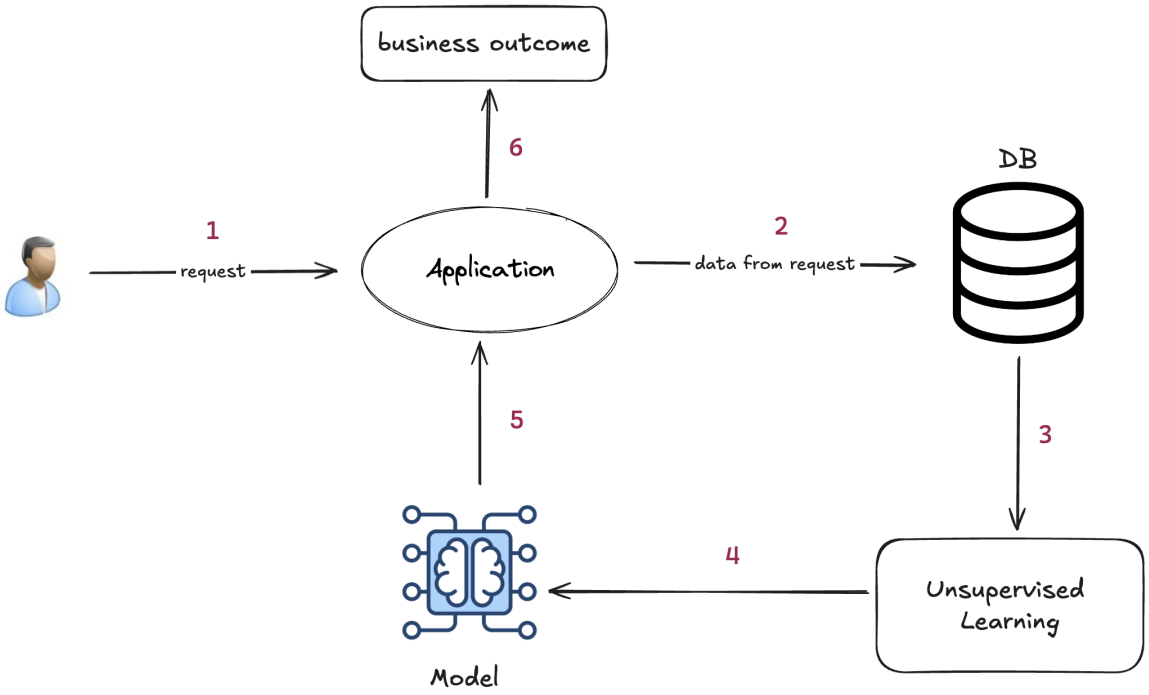

Figure 1: Typical application architecture that uses an anomaly detection technique

- `Q2 (Efficiency)`: The software that runs the AD model needs to operate very quickly — think about the urgency of a credit card transaction — where every nanosecond matters. Can we reduce the inference time?
- `Q3 (Interoperability)`: Integrating the AD model into an application can be challenging due to various practical considerations, such as where the model is stored and how it is accessed. Can we simplify the integration process with the application?

In this paper, we explore these questions. Our main contributions can be summarized as follows:

- We analyze the practical limitations of AD models in real-world applications and propose solutions to address these challenges.
- We implement our proposed solution in a real-world application, evaluate its effectiveness, and assess the associated trade-offs.

The paper is organized as follows: In Section 2, we provide a formal definition of the problem. Section 3 outlines our approach to addressing this problem, discussing the rationale behind why our solution is expected to answer research questions Q1-Q3, as well as the potential trade-offs involved. In Section 4, we present our experimental evaluation based on the implementation of our solution in a real-world application. Section 5 reviews related work, and Section 6 concludes the paper.

2. Problem Statement

We consider the architecture illustrated in Figure 1 and assume that the data extracted from user requests (step 2) is in a relational format. Specifically, we assume that each user request r is transformed into a tuple t by a function f. The tuple $t = f(r)$ consists of pairs $\{(c_i, v) \mid c_i \in C, v \in R, i \in \{1, \ldots, n\}\}$, where C represents a set of n column names and R denotes a set of real numbers. These tuples are stored in a database D, which can be a relational database or a data lake.

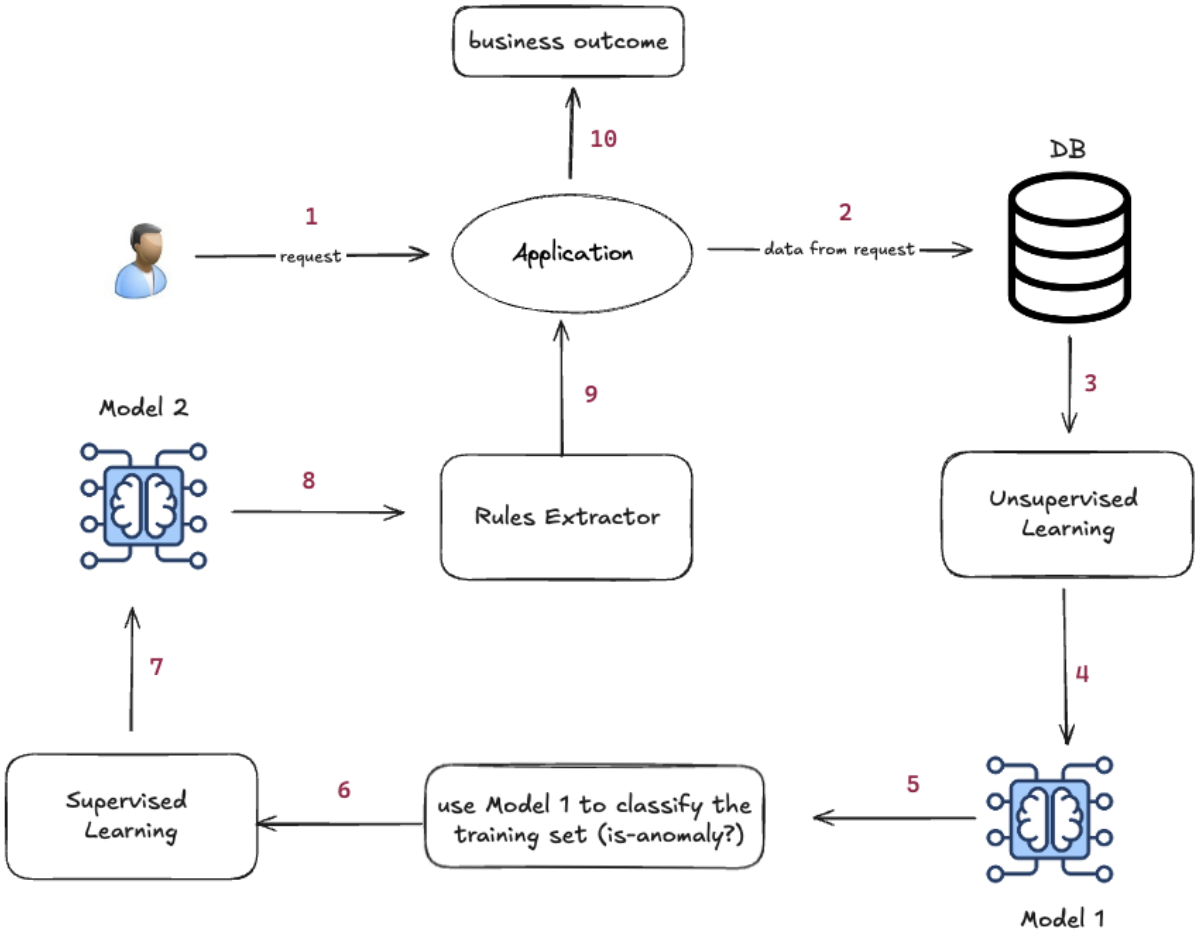

Figure 2: Our approach

In step 3, we apply an unsupervised AD algorithm (e.g., Local Outlier Factor [5] or Isolation Forest [6]) to a subset of columns in the database to build an AD model, denoted as M, that will be used in the application (step 5). It is possible to create multiple models during this step, such as distinct models for each customer or separate models for different features. Different features are defined by different subsets of columns. For example, we can have model M_1 trained on columns c_1, c_3, c_7 and model M_2 trained on columns c_2, c_7, c_8, c_9. For each AD model M_i, we assign a projection function f_i that takes the tuple $f(r)$ and retains only the relevant columns for the model M_i. A model M_i is a function that takes as an argument $f_i(f(r))$ and returns a boolean flag indicating whether the given request is considered an anomaly.

The models M_i are made accessible to the application (step 4), either by deploying them on a dedicated service (e.g., AWS SageMaker [7]) or by keeping them in the application's memory. For each user request r and model M_i, the application determines whether the request is an anomaly based on the response from $M_i(f_i(f(r)))$ and executes the corresponding business operation (step 6).

The goal of this paper is to address research questions Q1-Q3 within the framework of the presented system model.

3. Our Approach

Our approach enhances the flow shown in Figure 1 by expanding it to the one presented in Figure 2. Steps 1 to 4 remain unchanged. However, in step 5, instead of directly exposing the AD model (Model 1)

to the application, we use it to label our training set (which is the same training set used to train Model 1 in step 3 of Figure 1). For each tuple t, we add an additional boolean column labeled "is-anomaly," which indicates whether the specific tuple is an anomaly.

Formally, for each model M_i and each tuple $t \in D$, we build a new training set D', such that for each $t' \in D'$, the following holds:

$$t' = t \cup (\text{"}is - anomaly\text{"}, M_i(f_i(t)))$$

.

In step 6, we perform supervised learning using a Decision Tree on the labeled training set D' to create a model (Model 2) that predicts whether a given instance is an anomaly. As a result, in step 7, we have a decision tree model that makes anomaly predictions similar to those of Model 1, but in a more interpretable format.

In step 8, we apply Algorithm 1 to extract rules from Model 2. These rules are predicates in a Disjunctive Normal Form (DNF), which look like $(T_{11} \wedge T_{12} \wedge \dots) \vee (T_{21} \wedge T_{22} \dots) \cdots \vee (T_{m1} \wedge T_{m2} \dots)$. A term T_{ij} is a condition of type `<column op value>` (e.g., age > 40), where columns are taken from C, values from R, and $op \in \{\leq, >\}$.

The rules are then stored in the application's memory in step 9. In step 10, similar to step 6 in Figure 1, the application can use these rules to assess user requests and determine whether a specific request is an anomaly.

Algorithm 1 Extracting DNF Rules from a Decision Tree

```
1:  procedure ExtractRules(root)
2:      rules ← {}
3:      FindPaths(root, "", rules)
4:      return ⋁_{r∈rules} r
5:  end procedure
6:  procedure FindPaths(node, path, rules)
7:      if node is a leaf then
8:          if node.label = true then
9:              rules ← rules ∪ path
10:         end if
11:         return
12:     end if
13:     leftCond ← node.feature ≤ node.threshold
14:     leftPath ← Append(path, leftCond)
15:     FindPaths(node.left, leftPath, rules)
16:     rightCond ← node.feature > node.threshold
17:     rightPath ← Append(path, rightCond)
18:     FindPaths(node.right, rightPath, rules)
19: end procedure
20: procedure Append(path, cond)
21:     if path is empty then
22:         return cond
23:     else
24:         return path ∧ cond
25:     end if
26: end procedure
```

Algorithm 1 traverses the decision tree from the root to the leaves. At each internal node, it records the split condition (using $\leq$ for the left branch and $>$ for the right branch) while building the current

path. When it reaches a leaf labeled "true," which indicates an anomaly, it adds the conjunction of conditions for that path to the rule set.

The final output consists of all these paths combined with OR, resulting in a DNF formula. The running time of this algorithm is $O(L \times d)$, where L represents the number of leaves and d denotes the tree depth.

We will now briefly review research questions Q1-Q3 and discuss how our approach addresses them.

- `Q1 (Explainability)`: Unlike the baseline approach, where the model functions as a black box, our method provides clear, human-readable logic for anomaly detection decisions (e.g., age > 40 AND $P_2 \leq 76$). As a result, the explainability of our method is significantly enhanced.
- `Q2 (Efficiency)`: We will present an efficiency analysis based on real-life tests in the following section. The intuition behind why our approach may be faster lies in the fact that, instead of complex model inference, we perform only basic operations - mainly number comparisons and boolean expression evaluations.
- `Q3 (Interoperability)`: Our approach simplifies integration with applications compared to the baseline. First, unlike the machine learning (ML) model, our rules occupy minimal memory space, making them easy to store and manage. Second, our rules can be interpreted across any programming language, including even esoteric ones , since they rely on fundamental primitives and do not require a special format. In contrast, accessing ML models typically requires mainstream programming languages.

The advantages mentioned above come with certain drawbacks. Two main issues with our approach compared to the baseline are as follows:

1. There is an overhead in steps 5-7 of Figure 2 due to the processes involved in labeling the training set, supervised learning, and rule extraction. However, since this overhead occurs only occasionally (during retraining), its impact is not significant.
2. Our final model (Model 2) is designed to predict the outcomes of the original model (Model 1). Obviously, the prediction accuracy is unlikely to reach 100%, meaning our rules may not be as precise as those of the original model. This trade-off is the primary drawback of our approach, as we prioritize explainability, efficiency, and interoperability over accuracy.

4. Real-life Evaluation

We have implemented our approach in a real-life application that detects anomalous data points. The overall system architecture follows the flow in Figure 2. This application has been running in production mode for over six months. In this section, we will present our evaluation results while keeping internal application details confidential.

Our system follows the steps presented in Figure 2, with the following nuances:

- `Step 1`: The application receives billions of requests containing hundreds of different attributes.
- `Step 2`: Each request is stored in a relational format within Parquet files located in the S3 data lake.
- `Step 3`: A Spark MLlib job running on an AWS EMR cluster processes the data in the data lake to create Isolation Forest models. Five types of models are generated (identified by the letters A to E), differing in both the records and columns used for training.
- `Step 4`: The models created in the previous step are stored in ONNX format [8] in S3.
- `Step 5`: Models from the previous step are used to add "is-anomaly" labels to the training set for each model.
- `Step 6`: Another Spark MLlib job running on the AWS EMR cluster creates a Decision Tree model from the labeled training sets.

- `Step 7`: The Decision Tree model from the previous step is kept in the memory of the Spark cluster.
- `Step 8`: The same Spark cluster runs Algorithm 1 to processes the Decision Tree models and generate DNF rules.
- `Step 9`: The rules are stored in S3 in JSON format and exposed to the application. A real-life example of the rules, using anonymized column names, is shown in Listing 1.

Listing 1: DNF rules example

```
{
  "or": [
    {
      "and": [
        {"field": "c1", "value": 90, "operator": "<="},
        {"field": "c2", "value": 506, "operator": "<="},
        {"field": "c1", "value": 22, "operator": ">"},
        {"field": "c2", "value": 120, "operator": ">"}
      ]
    },
    {"and": [{"field": "c1", "value": 90, "operator": ">"}]}
  ]
}
```

We will now compare our approach with the baseline described in Figure 1 across the following dimensions:

- `Storage`: Comparing the storage size of the rules (in JSON format) with that of the Isolation Forest model (iForest) in ONNX format.
- `Performance`: Comparing the inference performance of the rules against the Isolation Forest model.
- `Accuracy`: Analyzing the accuracy degradation of our rules in comparison to the Isolation Forest model.

4.1. Storage

The results of the storage evaluation are presented in Table 1. For each model, we provided the storage size and memory loading time for each case (rules in JSON format versus iForest in ONNX format).

Both the storage size and loading time were an order of magnitude smaller with our approach (using rules) compared to the baseline (using iForest) across all models.

Model Type	Rules Size (KB)	ONNX Size (KB)	Rules Load Time (sec)	iForest Load Time (sec)
A	46	533	0.0007	0.0267
B	37	830	0.0006	0.0375
C	40	477	0.0009	0.0254
D	26	642	0.0007	0.0151
E	51	492	0.0006	0.0271

Table 1
Comparison of Rules vs iForest Model (Storage)

4.2. Performance

The performance evaluation is summarized in Table 2. We assessed two key metrics for each model (iForest vs. Rules): offline processing overhead and inference time.

For the offline processing measurement, we recorded the time required to train the iForest model and compared it with the time taken by our approach, which includes labeling, decision tree training, and rules extraction - all combined.

For the inference evaluation, we used real traffic data, and the number of records for each model is indicated in the table under "Records Num." The inference time is calculated as the average time taken to make a prediction for a single record.

The results show that our rules inference method outperformed iForest inference across all models, achieving an overall average advantage of 33%. While offline processing overhead increased by approximately 50%, it's important to note that this overhead occurs only occasionally during retraining, making it relatively insignificant.

Model Type	Records Num	Rules Inference (ns)	iForest Inference (ns)	Rules Extraction (sec)	iForest Training (sec)
A	2,301,358	7,555	9,112	52	132
B	32,677,265	6,916	9,700	82	150
C	791,083	7,702	8,566	63	121
D	36,453,915	2,686	8,938	48	117
E	5,472,485	9,398	9,619	83	172

Table 2
Comparison of Rules vs iForest Model (Performance)

4.3. Accuracy

We evaluate the accuracy of our approach by comparing the prediction results of the rules generated by our method to the results produced by the iForest model. We use the same validation sets as those used in the inference evaluation in the previous section (4.2). Our analysis focuses on three key metrics: Precision (True Positives / (True Positives + False Positives)), Recall (True Positives / (True Positives + False Negatives)), and the F1 score (a balanced measure of Precision and Recall). The findings are summarized in Table 3.

The accuracy of the models varies, with an average F1 score of around 77%. However, it is important to note that many samples misclassified by the rules differed from the classifications made by the iForest model by only a small margin. This indicates that the practical accuracy may be better than what is shown in Table 3. For example, we found that approximately 87% of all misclassified points were within just 0.05 of the iForest threshold — the score that distinguishes between anomalies and valid points.

Model Type	Precision	Recall	F1
A	84.73%	66.87%	74.75%
B	69.47%	69.58%	69.53%
C	91.09%	72.92%	81.00%
D	77.83%	77.58%	77.70%
E	72.84%	93.08%	81.72%

Table 3
Comparison of Rules vs iForest Model (Accuracy)

5. Related Work

AD has been an active area of research for decades, with foundational studies beginning as far back as the 1960s [9]. Today, numerous AD algorithms exist, including Isolation Forest [6], Local Outlier Factor [5], One-Class Support Vector Machine [10], and K-Means-based [11]. These algorithms are applied across various domains such as finance [1], cybersecurity [2], manufacturing [12], and healthcare [3]. Comprehensive reviews of AD techniques can be found in [4], [13], and [14]. We do not compare our system with existing AD algorithms as our primary contribution is not a new detection algorithm but a

deployment pipeline. This pipeline can wrap any unsupervised detector, including LOF, iForest, or deep models, with equal effect.

Explainability in Anomaly Detection Carletti et al. [15] propose DIFFI, a depth based feature importance measure designed specifically for IForest. It ranks each feature's contribution to the anomaly score without retraining the model. Chawla et al. [16] introduce a cluster based autoencoder for network anomaly detection and integrate SHAP values [16] to provide feature level explanations for each prediction in real world telecom systems. While these approaches explain individual anomaly decisions effectively, they still rely on the original model during inference. This means that inference latency, storage requirements, and the dependency on a ML runtime remain unchanged. Our method is fundamentally different. Instead of adding explanations to a black box model, we replace the model entirely with globally interpretable DNF rules. These rules are self explanatory by design, language agnostic, and require only simple arithmetic comparisons to evaluate.

Hybrid Supervised and Unsupervised Approaches. Several recent studies combine unsupervised and supervised learning for anomaly detection. Shanaa and Abdallah [17] propose a hybrid framework that uses an autoencoder trained on normal transactions together with an XGBoost classifier. Their method achieves state of the art F1 scores on a public credit card fraud benchmark. The design in [17] is similar to ours because it also uses an unsupervised component to generate training signals for a supervised model. However, there are three main differences. First, [17] focuses primarily on detection accuracy, while we prioritise explainability, efficiency, and interoperability. Second, in [17], both the autoencoder and the classifier must run during serving, whereas in our approach everything is reduced to a compact set of rules. Third, we evaluated our method on a real production system with billions of actual requests rather than only on a public benchmark dataset, which makes our results more realistic and practical.

6. Conclusion

In this paper, we addressed three critical research questions — explainability (Q1), efficiency (Q2), and interoperability (Q3) — that often make the practical application of AD in real-world systems challenging. By combining unsupervised and supervised learning we transformed "black box" iForest models into interpretable DNF rules.

Our real-life evaluation, demonstrated significant practical benefits:

- `Efficiency:` The rule-based inference method outperformed the baseline iForest inference by an average of 33%, while storage requirements and memory loading times were reduced by an order of magnitude.
- `Interoperability:` By storing rules in JSON format, we eliminated the need for specialized ML environments, allowing the logic to be interpreted across any programming language using fundamental primitives.
- `Explainability:` The transition from complex ML models to human-readable predicates provides users with clear reasoning for anomaly detection decisions.

While we observed an accuracy trade-off, with an average F1 score of approximately 77%, the majority of misclassified points were within a narrow margin (0.05) of the original model's threshold. We believe this trade-off is justified for applications where speed and transparency are prioritized over absolute precision. Future work could focus on further narrowing this accuracy gap while maintaining the efficiency of the rule-extraction framework.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly in order to: Improve writing style, and Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] E. L. Paula, M. Ladeira, R. N. Carvalho, T. Marzagao, Deep learning anomaly detection as support fraud investigation in brazilian exports and anti-money laundering, in: 2016 15th ieee international conference on machine learning and applications (icmla), IEEE, 2016, pp. 954–960.

[2] A. K. Ghosh, J. Wanken, F. Charron, Detecting anomalous and unknown intrusions against programs, in: Proceedings 14th annual computer security applications conference (Cat. No. 98Ex217), IEEE, 1998, pp. 259–267.

[3] W.-K. Wong, A. W. Moore, G. F. Cooper, M. M. Wagner, Bayesian network anomaly pattern detection for disease outbreaks, in: Proceedings of the 20th international conference on machine learning (ICML-03), 2003, pp. 808–815.

[4] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM computing surveys (CSUR) 41 (2009) 1–58.

[5] M. M. Breunig, H.-P. Kriegel, R. T. Ng, J. Sander, Lof: identifying density-based local outliers, in: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, 2000, pp. 93–104.

[6] F. T. Liu, K. M. Ting, Z.-H. Zhou, Isolation-based anomaly detection, ACM Transactions on Knowledge Discovery from Data (TKDD) 6 (2012) 1–39.

[7] E. Liberty, Z. Karnin, B. Xiang, L. Rouesnel, B. Coskun, R. Nallapati, J. Delgado, A. Sadoughi, Y. Astashonok, P. Das, et al., Elastic machine learning algorithms in amazon sagemaker, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, pp. 731–737.

[8] ONNX, Open neural network exchange, 2025. URL: https://github.com/onnx/onnx.

[9] F. E. Grubbs, Procedures for detecting outlying observations in samples, Technometrics 11 (1969) 1–21.

[10] M. Hejazi, Y. P. Singh, One-class support vector machines approach to anomaly detection, Applied Artificial Intelligence 27 (2013) 351–366.

[11] M. Elahi, K. Li, W. Nisar, X. Lv, H. Wang, Efficient clustering-based outlier detection algorithm for dynamic data stream, in: 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery, volume 5, IEEE, 2008, pp. 298–304.

[12] E. Keogh, J. Lin, S.-H. Lee, H. V. Herle, Finding the most unusual time series subsequence: algorithms and applications, Knowledge and Information Systems 11 (2007) 1–27.

[13] A. Boukerche, L. Zheng, O. Alfandi, Outlier detection: Methods, models, and classification, ACM Computing Surveys (CSUR) 53 (2020) 1–37.

[14] G. Pang, C. Shen, L. Cao, A. V. D. Hengel, Deep learning for anomaly detection: A review, ACM computing surveys (CSUR) 54 (2021) 1–38.

[15] M. Carletti, M. Terzi, G. A. Susto, Interpretable anomaly detection with DIFFI: depth-based feature importance of Isolation Forest, Engineering Applications of Artificial Intelligence 116 (2023) 105730.

[16] A. Chawla, P. Jacob, P. Farrell, E. Aumayr, S. Fallon, Towards interpretable anomaly detection: unsupervised deep neural network approach using feedback loop, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), IEEE, 2022, pp. 1–9.

[17] M. Shanaa, S. Abdallah, A hybrid anomaly detection framework combining supervised and unsupervised learning for credit card fraud detection, F1000Research 14 (2025).