Evaluating Machine Learning Algorithms for Anomaly Detection in Industrial Engines on Edge Devices

Lucas Araújo[1], **Sérgio Chevtchenko**[2], **Danilo Araújo**[1], **Ermeson Andrade**[1]

[1] Universidade Federal Rural de Pernambuco (UFRPE)

{lucas.edson, danilo.araujo, ermeson.andrade}@ufrpe.br,

[2]Western Sydney University

s.chevtchenko@westernsydney.edu.au

Abstract. *Industrial automation and the Internet of Things (IoT) are rapidly evolving, highlighting the need for efficient anomaly detection systems to prevent failures and reduce maintenance and operational costs. This study evaluates the feasibility of running machine learning-based anomaly detection models on edge devices like the Raspberry Pi. Using real sensor data from industrial engines with induced faults, we tested three unsupervised algorithms: LOF, OCSVM, and IF. Metrics such as sensitivity, specificity, inference time, and resource usage were analyzed under different loads. The results show that accurate and low-latency detection is possible on resource-constrained devices, supporting the development of cost-effective and autonomous monitoring systems for industrial environments.*

1. Introduction

In the dynamic landscape of industrial automation and the Internet of Things (IoT), the demand for anomaly detection systems aimed at fault prevention and cost reduction has become increasingly critical. As IoT systems become more embedded in industrial settings, the need for reliable methods to detect faults early becomes essential. Early identification of anomalies helps avoid equipment failures, supports operational continuity, and reduces maintenance costs. Implementing such systems on resource-constrained devices is particularly relevant, given their widespread use in IoT ecosystems. On edge devices, anomaly detection enables low-latency responses and leverages local processing to identify issues in real time, preventing disruptions and promoting continuous process optimization [Chevtchenko et al. 2023a, Mukherjee et al. 2023].

Machine learning (ML) models play a key role in anomaly detection for industrial equipment via IoT devices. Practical applications include early fault detection in engines, pumps, and other critical equipment, allowing for preventive maintenance and cost savings. Their importance becomes even more evident when considering the need for detailed anomaly evaluation over time, which is crucial for continuous machine condition monitoring [Schmidl et al. 2022]. Furthermore, anomaly detection and diagnostics can be tailored to complex industrial scenarios, such as automated production lines and quality control systems, where multiple variables interact in unpredictable ways [Garg et al. 2022]. However, to enable real-time, low-latency detection, these analyses must be performed locally on edge devices, which have limited computational capacity. Therefore, assessing the feasibility of running such models locally is essential.

The literature has explored the potential of ML algorithms in anomaly detection. For instance, [Chevtchenko et al. 2023b] demonstrated the effectiveness of predictive maintenance models based on anomaly detection in induction engines. They employed machine learning algorithms to analyze cloud-based data generated in real time by an IoT device. Similarly, [Ghazal et al. 2020] proposed a cloud-based architecture that combines AI and the Internet of Robotic Things (IoRT) for detecting anomalies in industrial environments. The work by [Mian et al. 2023] demonstrated the use of deep learning for anomaly detection in rotating machinery, although their approach focuses on simulation rather than real embedded devices. These are just a few examples among many studies that highlight the relevance of machine learning for anomaly detection. However, most existing approaches rely on cloud infrastructure, which may not be suitable for scenarios requiring real-time processing or operating under connectivity constraints.

To address this gap, we adopt an approach focused on evaluating anomaly detection models directly on low-cost edge devices. We emphasize the use of the Raspberry Pi, given its widespread adoption and affordability. Our goal is to examine how these models can be effectively adapted and deployed in constrained environments. Specifically, we investigate the performance of three unsupervised algorithms: One-Class Support Vector Machine (OCSVM), Isolation Forest (IF), and Local Outlier Factor (LOF), applied to anomaly detection in industrial engines. The analysis considers metrics such as accuracy, inference time, and resource usage. This evaluation aims to demonstrate the feasibility of local execution and reinforce the potential of edge-based solutions for real-time fault detection in industrial applications.

The remainder of this paper is organized as follows: Section 2 details the experimental methods, including the proposed anomaly detection approaches, experimental setup, and implementation details. Section 3 presents the experimental results, providing an in-depth analysis of the system's performance and effectiveness under varying loads. Finally, Section 4 concludes the paper by summarizing the main findings and discussing future research directions in this field.

2. Experimental Methods

This section details the experimental procedures used to implement and evaluate machine learning-based anomaly detection systems in edge computing environments.

2.1. Data Structure

In this study, we used data from three-phase induction engines monitored by connected IoT sensors, as described by [Chevtchenko et al. 2023a]. These sensors collected temperature, accelerometer, gyroscope, and microphone data. To generate anomalous data, three distinct faults were artificially induced: unbalanced load (engine 1), damaged bearings (engine 2), and stator short-circuit (engine 3). Data preprocessing followed methods from Chevtchenko et al.'s prior work, utilizing Fast Fourier Transform (FFT) and Discrete Wavelet Transform (DWT) for noise reduction and feature extraction. Principal Component Analysis (PCA) was then applied for dimensionality reduction, optimizing model performance and reducing computational complexity.

2.2. Dataset Structure

The dataset used in this study consists of sensor readings collected from three industrial engines, totaling 6,000 samples. For each engine, the data were divided into three subsets: training, validation, and testing. Each subset includes both normal operation and fault condition data, as described below:

- **Training**: 500 normal samples per engine.
- **Validation**: 500 normal and 500 anomalous samples per engine, totaling 1,000 samples. These samples were not used directly for model tuning, as the primary analysis focused on evaluating performance using the test set.
- **Testing**: 250 normal samples and 250 anomalous samples per engine, totaling 500 test samples.

Thus, for each engine, we used 500 (training) + 1,000 (validation) + 500 (testing) = 2,000 samples. Considering three engines, the complete dataset comprises 3 × 2,000 = 6,000 samples.

2.3. Anomaly Detection Models

Three anomaly detection models were employed in this study: LOF, OCSVM, and IF. These models were executed on a Raspberry Pi 4, and the runtime data were collected in that environment. The selection of specific hyperparameters aimed at optimizing model performance was based on the previous work by [Chevtchenko et al. 2023a]. The parameter settings for each model are described below.

2.3.1. LOF Algorithm

The LOF algorithm was employed to perform a comparative analysis between the local density of a data point and that of its neighbors. Points with significantly lower density compared to their surrounding neighbors are classified as anomalies. The LOF model proves to be highly relevant for dataset analysis, particularly for optimizing the detection of noise and vibration signals. In our configuration, 200 neighbors were considered for the analysis, using the Minkowski distance metric with a parameter $p = 2$, which corresponds to the Euclidean distance [Breunig et al. 2000].

2.3.2. OCSVM Algorithm

OCSVM is based on the Support Vector Machines (SVM) technique, in which normal data are enclosed within a single class. By modeling only the normal behavior, the algorithm can detect anomalies as data points that significantly deviate from this learned pattern. We used the Radial Basis Function (RBF) kernel, one of the hyperparameters of OCSVM, due to its effectiveness in capturing nonlinear patterns. In the OCSVM configuration, the ν (nu) parameter was set to 0.1, an empirically adjusted value that controls the proportion of anomalies and the number of support vectors. The γ (gamma) parameter was set to `scale`, which determines the scale of γ. The model tolerance was configured as 1×10^{-3}, which defines the stopping criterion for adjusting the decision boundary [Schölkopf et al. 2001].

2.3.3. IF Algorithm

The IF algorithm isolates anomalies by creating random partitions of the dataset, or decision trees, to separate anomalous points from those considered normal. This technique is efficient in data processing and is especially suitable for resource-constrained environments. In the Isolation Forest configuration, we used 100 estimators (trees). The expected anomaly proportion was set to 0.1. The maximum sample size was set to `auto`, meaning it is automatically determined. No replacement was used in the samples (bootstrap was set to `False`), and the model was configured to utilize multiple processing cores (`n_jobs` set to -1). Additionally, the maximum proportion of features considered was set to 0.9. These settings enhance model efficiency and make it suitable for environments with limited resources, such as the Raspberry Pi [Liu et al. 2008].

2.4. Model Evaluation

The evaluation of the anomaly detection models was conducted based on multiple criteria to ensure that the models not only accurately detected anomalies but also operated efficiently in an edge computing environment. The main evaluation criteria are as follows:

- **Recall (Sensitivity)**: Recall indicates the model's ability to correctly identify anomalies, measuring the proportion of true positives relative to the total number of actual anomalies. This metric is particularly important for assessing how well the model detects anomalous cases [Powers 2011].
- **Specificity**: Specificity measures the model's ability to correctly identify normal operations, i.e., the proportion of true negatives among all normal instances. This criterion is useful for evaluating the model's effectiveness in reducing false alarms [Powers 2011].
- **Inference Time**: Inference time evaluates how quickly the model can process inputs and provide a response. This metric is critical in edge environments where real-time decisions are required. A low inference time ensures that the system can promptly respond to detected anomalies, minimizing potential damage or interruptions [Dutta and Mukhopadhyay 2020].
- **Resource Efficiency**: This criterion assesses the consumption of computational resources, including CPU and memory. Models that use resources efficiently are better suited for edge devices with hardware constraints, such as the Raspberry Pi [Foundation 2019].

2.5. Platform

To execute the anomaly detection models, we adopted an edge computing platform using a Raspberry Pi 4 Model B. The choice of the Raspberry Pi reflects its widespread use in IoT applications, due to its low cost, high efficiency, and sufficient processing capabilities for real-time monitoring tasks. The Raspberry Pi 4 provides a robust performance-to-power ratio, making it ideal for scenarios where resources are limited and energy efficiency is crucial [Mian et al. 2023].

2.6. Experiment Configuration and Execution

The experimental infrastructure was configured with the following components, according to the hardware specifications presented in Table 1:

- **Client Computer**: Used to simulate the data transmission to the server.
- **Raspberry Pi 4 Model B (Server)**: Used to process the received data.
- **Communication Protocol**: MQTT (Message Queuing Telemetry Transport), chosen for its lightweight and efficient characteristics in IoT scenarios.

Table 1. Hardware specifications of the environment used.

Environment	Details
Client, Dell Inspiron 15	Processor: Intel Core i5-1135G7, 2.40GHz RAM: 16 GB (15.7 GB usable) OS: Windows 64-bit, x64
Server, Raspberry Pi 4 Model B	Processor: BCM2711, 1.5 GHz (4 cores) RAM: 4 GB OS: Raspberry Pi OS Buster

2.7. Experimental Execution

During the experimental execution, the following steps were carried out: The data captured by the IoT sensors were initially unpacked in the client environment. Subsequently, these data were transmitted to the server via the MQTT protocol, with each MQTT message containing a single data sample. On the Raspberry Pi server, a Python-based API was developed to receive the MQTT messages and process the data using the models adopted in this study.

Throughout the experiments, the Raspberry Pi was monitored for metrics such as resource usage efficiency, recall, and response time. Additionally, the system's behavior was evaluated under different request load intervals (0.1, 0.5, and 1.0 seconds) [Doe and Smith 2023]. These load intervals represent the time between successive MQTT messages sent to the server, simulating different operational scenarios found in real-world industrial environments. For example, a load of 0.1 seconds means that the system receives a new data sample every 100 milliseconds, which corresponds to a high-frequency monitoring scenario. Such a rate might be expected in critical systems with fast-changing signals, like high-speed rotating machinery or precision manufacturing lines. A load of 0.5 seconds represents a moderate monitoring frequency, suitable for systems where conditions change rapidly but not as aggressively. Finally, a 1.0-second interval simulates a low-frequency monitoring scenario, often found in more stable industrial processes, such as temperature or vibration tracking in non-critical components. These different rates were used to evaluate how the models and the edge infrastructure respond under varying data volumes and time constraints.

To collect detailed system metrics during model execution, the Python library `psutil` was employed. This tool is widely recognized for its versatility and ability to monitor and evaluate system performance in real time. Using `psutil`, detailed data on CPU usage, memory consumption, disk activity, and network performance were collected, providing critical insights into the Raspberry Pi server's behavior during model testing [Mudaliar and Sivakumar 2020]. Furthermore, for the visualization and analysis of the collected data, plotting libraries such as `matplotlib` [Hunter 2007] and `seaborn` [Waskom 2021] were used, in conjunction with `pandas` [McKinney 2010] for data manipulation and analysis.

3. Results

The efficiency of anomaly detection models in edge environments was evaluated on the Raspberry Pi platform, considering accuracy, computational resource usage, and inference time. The experiments were conducted with the device positioned close to the monitored system, minimizing communication latency and optimizing real-time processing. The models and datasets are available in the online repository[1].

3.1. Model Performance

The obtained results were grouped and analyzed across three distinct engines, allowing a comprehensive evaluation of model behavior under different operational conditions. Each model was executed 10 times for each engine to ensure a more robust and reliable assessment of the metrics. Table 2 presents the mean and standard deviation of sensitivity, specificity, and inference time, enabling a detailed comparison between the algorithms.

Table 2. Model Evaluation by Algorithm and Engine (mean ± standard deviation).

Algorithm	Metric	Engine 1	Engine 2	Engine 3
LOF	Sensitivity	0.643 ± 0.000	0.999 ± 0.000	0.615 ± 0.000
LOF	Specificity	0.931 ± 0.000	0.931 ± 0.000	0.803 ± 0.000
LOF	Inference Time (s)	0.364 ± 0.002	0.366 ± 0.004	0.367 ± 0.005
OCSVM	Sensitivity	0.779 ± 0.000	0.923 ± 0.000	0.999 ± 0.000
OCSVM	Specificity	0.967 ± 0.000	0.947 ± 0.000	0.519 ± 0.000
OCSVM	Inference Time (s)	0.288 ± 0.001	0.289 ± 0.003	0.289 ± 0.003
IF	Sensitivity	0.868 ± 0.020	0.999 ± 0.000	0.999 ± 0.000
IF	Specificity	0.854 ± 0.005	0.679 ± 0.023	0.413 ± 0.007
IF	Inference Time (s)	0.710 ± 0.007	0.708 ± 0.009	0.712 ± 0.006

In the case of the LOF algorithm, very high sensitivity was observed for engine 2 (0.999 ± 0.000), contrasting with lower values for engines 1 (0.643) and 3 (0.615). Specificity was high and stable for engines 1 and 2 (0.931), but showed a decrease for engine 3 (0.803). Inference time remained consistent across all engines, ranging from 0.364 to 0.367 seconds, making it feasible for real-time applications. The OCSVM algorithm demonstrated high sensitivity across all three engines, with a highlight on engine 3 (0.999). However, it exhibited a significant drop in specificity for the same engine (0.519), indicating a higher tendency for false positives. This behavior suggests that OCSVM may be more sensitive to noise or atypical patterns, especially in less stable environments. Its inference time was the lowest among the tested algorithms, consistently below 0.29 seconds, making it a strong candidate for systems with strict time constraints.

The IF algorithm, known for its stochastic approach based on random partitioning of the data, achieved consistently high sensitivity, particularly for engines 2 and 3 (0.999),

[1]`https://www.dropbox.com/scl/fo/7u5u7e3fr12iseuqkf5uz/AFZ18VlVgp_spY_PgTThIVY?rlkey=vsfap2y6paezst=uccwruew&dl=0`

with slightly lower performance for engine 1 (0.868 ± 0.020), accompanied by notable variability as reflected in the standard deviation. This variability is expected, as IF relies on randomly constructed isolation trees, introducing a non-deterministic behavior in the results. Therefore, multiple executions are essential to assess average performance and its stability. On the other hand, IF's specificity was lower than that of the other models, especially for engine 3 (0.413), indicating a higher false positive rate. Furthermore, its inference time was the highest among the models, exceeding 0.71 seconds in all executions, which may hinder its applicability in real-time systems.

In summary, the results highlight the unique characteristics of each algorithm. OCSVM stands out for its balanced performance between sensitivity and execution time, despite some variability in specificity. LOF offers a good balance between sensitivity and specificity, especially for engine 2, with stable and low inference times. IF, in turn, excels in detection capability (high sensitivity), but at the cost of higher computational demand and lower specificity — traits inherent to its stochastic nature and tendency to over-identify anomalies. Therefore, the ideal algorithm depends on the specific requirements of the application environment. For scenarios where maximum fault detection is prioritized (with tolerance for false positives and longer response time), IF is a robust choice. Conversely, for applications requiring fast response and greater balance between detection and reliability, OCSVM and LOF are more suitable alternatives, with OCSVM being preferable in resource-constrained environments.

When comparing the results of this study with those of [Chevtchenko et al. 2023a], it is notable that, despite using the same dataset and a similar feature extraction methodology, our fully embedded approach achieved comparable sensitivity and specificity performance. While their cloud-based solution reported shorter inference times (e.g., 0.43 ms with OC-SVM), our models, executed entirely on an edge device, achieved inference times below 0.29 seconds with OCSVM and LOF. Although slightly slower, our approach demonstrates that it is feasible to deploy these models locally without relying on cloud infrastructure, which is crucial for applications with limited connectivity, real-time processing needs, or strict data privacy requirements.

3.2. Resource Usage Efficiency

For a more in-depth analysis, we compared the performance of the algorithms across the three engines under different loads. Figures 1, 2, and 3 present a detailed analysis of CPU and memory usage for the LOF, OCSVM, and IF algorithms under different operational conditions. Comparing computational resource usage is critical for evaluating the efficiency of real-time systems, especially on embedded devices such as the Raspberry Pi.

3.2.1. LOF Algorithm

As shown in Figures 1, the three engines exhibited similar resource usage patterns, even under varying load levels. However, a specific limitation was observed when testing under a load of 0.1, where the system crashed and terminated the programs due to hardware constraints. This incident highlights the importance of considering available resource capacity when implementing predictive maintenance solutions, especially in environments with limited computational power.

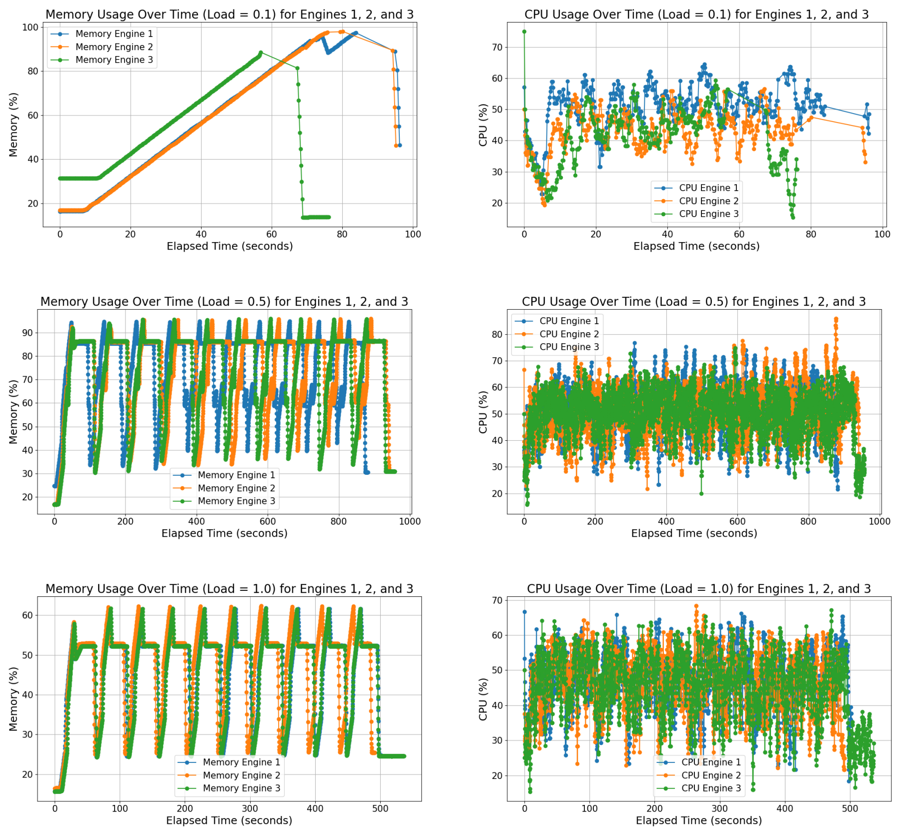

Figure 1. CPU and Memory Usage Comparison: LOF.

Moreover, under loads of 0.5 and 1.0, the behavior of the LOF algorithm remained relatively stable, although a significant increase in both memory and CPU usage was recorded. This suggests that while LOF is effective in detecting anomalies in high-dimensional datasets, its performance may be impacted by infrastructure limitations, particularly in resource-constrained systems. These observations indicate that code-level or underlying infrastructure optimizations may be necessary to ensure the algorithm continues to operate efficiently in more demanding scenarios.

3.2.2. IF Algorithm

Figure 2 illustrates CPU and memory usage for the Isolation Forest algorithm under different loads. Compared to LOF, IF shows a considerable increase in resource consumption, especially under loads of 0.5 and 1.0. This increase is related to the model's complexity, suggesting a limitation in its applicability to embedded systems with CPU and memory constraints. Furthermore, when testing with a load of 0.1, an improvement in CPU and memory usage was observed. However, the model still exhibited instability, with performance drops and occasional crashes. This behavior highlights the IF algorithm's sensitivity to small load variations, indicating that even in low-load scenarios, the model may be susceptible to hardware limitations. These findings reinforce the need for resource optimization when deploying models like IF in resource-constrained environments.

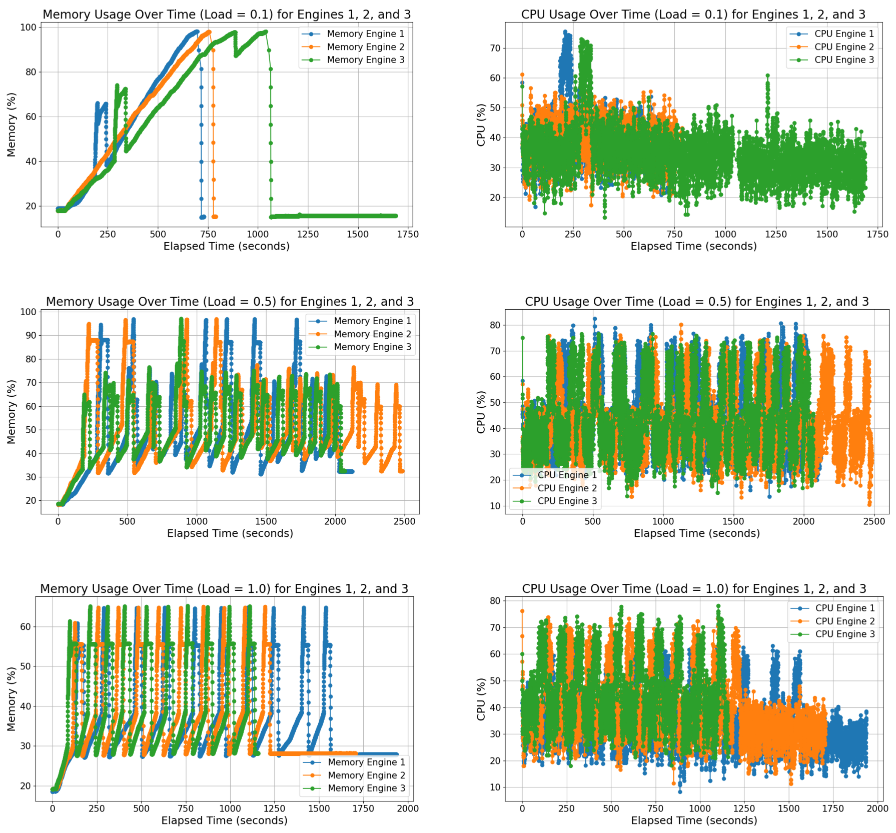

Figure 2. CPU and Memory Usage Comparison: IF.

3.2.3. OCSVM Algorithm

As shown in Figure 3, the OCSVM algorithm demonstrated lower resource consumption compared to IF, and memory usage similar to that of LOF. This behavior is particularly advantageous in scenarios where response time is critical, as OCSVM offers a good balance between detection performance and resource usage. Additionally, studies such as [Araújo-Filho 2018] suggest that although OCSVM may achieve higher accuracy in anomaly detection, IF can be more CPU-efficient under certain conditions.

When tested with a load of 0.1, OCSVM maintained optimized performance in terms of CPU and memory usage. However, similar to IF, the algorithm experienced occasional performance drops and instability during testing. This indicates that despite its lower resource usage, OCSVM is also sensitive to hardware limitations in high-load scenarios, highlighting the need for optimization to ensure stable operation in resource-constrained environments. Although OCSVM is more resource-efficient compared to the other algorithms, the literature indicates that fine-tuning is essential for each specific application to guarantee reliable performance in embedded systems [Molina 2022].

3.2.4. Conclusion of Resource Evaluation

Overall, the IF algorithm exhibited the highest computational resource consumption, especially under increased loads, while the OCSVM stood out for its lower inference time,

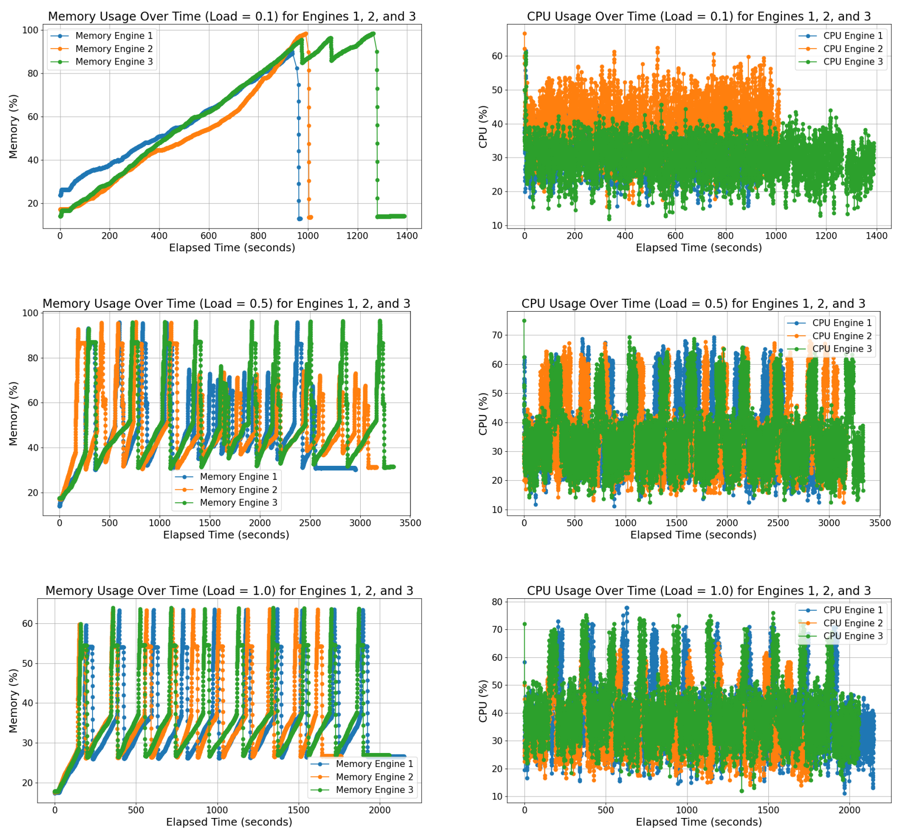

Figure 3. CPU and Memory Usage Comparison: OCSVM.

making it more suitable for real-time environments. The LOF offered an intermediate solution, with stable performance in terms of CPU and memory usage. These results suggest that the choice of the most appropriate algorithm depends on the time constraints and available resources within the application environment.

High-load intervals, which correspond to shorter gaps between requests, impose greater computational demands on the system, as the server must process a larger number of messages per unit of time. This leads to increased CPU and memory usage, which, on resource-constrained devices such as the Raspberry Pi, may result in crashes, performance degradation, or failures in algorithm execution. Therefore, the instability observed under the 0.1-second load is attributed to the cumulative processing overload caused by high request frequency, rather than the computational intensity of individual tasks.

When evaluating the performance of the algorithms across different scenarios, it was observed that IF tends to consume more memory and processing time, particularly as the data volume increases. This behavior results from its approach of constructing multiple decision trees, which introduces significant computational overhead. Conversely, OCSVM stood out for its rapid inference, which can be attributed to the use of a decision boundary learned during training. This makes OCSVM particularly advantageous in applications that demand fast responses, such as anomaly detection in critical systems or real-time data stream analysis.

LOF exhibited balanced performance, offering a compromise between resource consumption and anomaly detection accuracy. As a method based on local point density, LOF demonstrated efficiency in identifying anomalies with minimal impact on system resources, especially under moderate loads. This makes it a viable option for embedded systems where both detection performance and computational efficiency are important.Thus, the choice among these algorithms should be guided by the specific needs of the application. If minimizing inference time is the primary goal, OCSVM is an appropriate choice. If the application can tolerate higher resource usage in exchange for model robustness, IF may be suitable. LOF is a compelling alternative when a balance between resource consumption and detection accuracy is required.

4. Conclusion

This study investigated the implementation and evaluation of machine learning-based anomaly detection models on edge devices, focusing on the Raspberry Pi platform. The results demonstrate the technical feasibility of deploying models such as LOF, OCSVM, and IF in resource-constrained environments. Among the findings, OCSVM stood out for its low inference time and balanced performance, while LOF showed consistent behavior under moderate loads. IF achieved high sensitivity but at the cost of greater computational demand and reduced specificity. These trade-offs highlight that algorithm choice should be aligned with the specific requirements of each application scenario, such as latency tolerance, fault criticality, and resource availability.

The results reinforce the potential of low-cost edge computing as a viable alternative to cloud-based monitoring in industrial environments. When hardware limitations are considered, and the models are carefully selected, it is possible to design efficient and autonomous predictive maintenance systems. Future work may explore the scalability of this approach with larger datasets, investigate training-time optimizations, and evaluate the integration of semi-supervised or unsupervised techniques to enhance generalization across diverse operating conditions.

References

Araújo-Filho, J. G. (2018). Comparação de desempenho de algoritmos de detecção de anomalias na rede can. *Revista Brasileira de Engenharia Elétrica*, 36(1):1–10.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). Lof: identifying density-based local outliers. *ACM sigmod record*, 29(2):93–104.

Chevtchenko, S. F., Rocha, E. D. S., Dos Santos, M. C. M., Mota, R. L., Vieira, D. M., De Andrade, E. C., and De Araújo, D. R. B. (2023a). Anomaly detection in industrial machinery using iot devices and machine learning: A systematic mapping. *IEEE Access*, 11:128288–128305.

Chevtchenko, S. F., Santos, M., Vieira, D. M., Mota, R. L., Rocha, E., Cruz, B. V., Araújo, D., Andrade, E., et al. (2023b). Predictive maintenance model based on anomaly detection in induction motors: A machine learning approach using real-time iot data. *arXiv preprint arXiv:2310.14949*.

Doe, J. and Smith, J. (2023). Performance evaluation of raspberry pi for mqtt under different request loads. *Journal of IoT Applications*, 12(3):45–58.

Dutta, D. and Mukhopadhyay, D. (2020). *Edge AI: Machine Learning on Edge Devices*. O'Reilly Media, Sebastopol, CA, 1 edition.

Foundation, R. P. (2019). *Raspberry Pi 4 Model B: Technical Specifications*. `https://www.raspberrypi.org/`.

Garg, A., Zhang, W., Samaran, J., Savitha, R., and Foo, C.-S. (2022). An evaluation of anomaly detection and diagnosis in multivariate time series. *IEEE Transactions on Neural Networks and Learning Systems*, 33(6):2508–2517.

Ghazal, M., Basmaji, T., Yaghi, M., Alkhedher, M., Mahmoud, M., and El-Baz, A. S. (2020). Cloud-based monitoring of thermal anomalies in industrial environments using ai and the internet of robotic things. *Sensors*, 20(21):6348.

Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*, volume 9. IEEE.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE.

McKinney, W. (2010). Data structures for statistical computing in python. `https://pandas.pydata.org/`.

Mian, T., Choudhary, A., Fatima, S., and Panigrahi, B. (2023). Artificial intelligence of things based approach for anomaly detection in rotating machines. *Computers and Electrical Engineering*, 109:108760.

Molina, A. L. B. (2022). Weapon: Uma arquitetura de aprendizado não supervisionado para detecção de anomalias de comportamento de usuários. Master's thesis, Universidade de Brasília.

Mudaliar, M. D. and Sivakumar, N. (2020). Iot based real time energy monitoring system using raspberry pi. *Internet of Things*, 12:100292.

Mukherjee, I., Sahu, N. K., and Sahana, S. K. (2023). Simulation and modeling for anomaly detection in iot network using machine learning. *International Journal of Wireless Information Networks*, 30(2):173–189.

Powers, D. M. (2011). Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63.

Schmidl, S., Wenig, P., and Papenbrock, T. (2022). Anomaly detection in time series: a comprehensive evaluation. *Proceedings of the VLDB Endowment*, 15(9):1779–1797.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471.

Waskom, M. L. (2021). Seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.