



**HAL**  
open science

## Anomaly Detection in Edge Devices

Tommaso Zoppi, Andrea Ceccarelli, Andrea Bondavalli, Nicola Peditto,  
Maurizio Giacobbe, Antonio Puliafito

► **To cite this version:**

Tommaso Zoppi, Andrea Ceccarelli, Andrea Bondavalli, Nicola Peditto, Maurizio Giacobbe, et al..  
Anomaly Detection in Edge Devices. SAFECOMP 2023, Position Paper, Sep 2023, Toulouse, France.  
hal-04191783

**HAL Id: hal-04191783**

**<https://hal.science/hal-04191783>**

Submitted on 30 Aug 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Anomaly Detection in Edge Devices

Tommaso Zoppi, Andrea Ceccarelli, Andrea Bondavalli  
University of Florence, Department of Mathematics and  
Informatics, Viale Morgagni 65 – Florence (IT)  
{tommaso.zoppi, andrea.ceccarelli, andrea.bondavalli}@unifi.it

Nicola Peditto<sup>1</sup>, Maurizio Giacobbe<sup>1</sup>, Antonio Puliafito<sup>1,2</sup>  
<sup>1</sup>SmartMe.io, via Salita Lardereria, 98129 Messina (IT)  
<sup>2</sup>Univ. of Messina, Dept. of Engineering, 98166 Messina (IT)  
{nicola, maurizio, antonio}@smartme.io

**Abstract** — With the growing processing power of computing systems and the increasing availability of massive datasets, machine learning algorithms have led to major breakthroughs in many different areas. This applies also to resource-constrained IoT and edge devices, which can benefit of relatively small - but smart - local anomaly detection tasks to detect failures and attacks. This paper overviews the process we are following to equip small devices with anomaly detection capabilities to make them self-aware of their health state and take appropriate countermeasures. Our process is applied to a Linux-based devices named ARANCINO, which has already been successfully used in several smart cities applications.

**Keywords** - *anomaly detection, failures, IoT, arancino, monitoring, edge computing.*

## I. BRINGING ANOMALY DETECTION TO THE EDGE

Edge learning refers to the deployment of Machine Learning (ML) algorithms at the network edge [8]. The key motivation of pushing learning toward the edge is to perform on-site preprocessing and filtering of data, and also to provide edge devices with sophisticated yet lightweight means to optimize their performance. However, bringing ML on the edge is far from trivial and comes with many potential issues and limitations [5], [6], [7]. Whereas the vast majority of studies on ML rely on lab setups for which we assume the availability of huge server farms, and accelerators as GPUs, deploying ML algorithms in the wild comes with obvious concerns. It is required a dedicated methodology to collect data, choose appropriate ML algorithms, train and deploy them on devices.

This study explores how to bring ML algorithms to edge devices and make them work as anomaly detectors to suspect failures. Noteworthy, while many attack datasets are available and can be used as reference, failure datasets for anomaly detection on edge devices are few or none.

Ultimately, the goal is to deploy a self-aware, or self-checking device that is able to seek for potential anomalies that are symptom of failures, and react accordingly.

## II. DESIGNING SELF-AWARE EDGE DEVICES

Our methodology for deploying anomaly detectors that suits the specific characteristics of edge devices relies on the following 4 main steps.

- S1. Create a fault model that covers most of the common software faults in Linux-based IoT devices.
- S2. Create a monitoring system that fits our case study but also applies to similar devices.
- S3. Perform fault injection campaigns in which we monitor the behavior of the target device under normal operating conditions and when faults are injected.
- S4. Use the collected data to train anomaly detectors that can then be deployed in the target device to monitor their detection and timing performance.

We provide the main ideas that guided our methodology. The device we are using for implementation is the ARANCINO [13] devices.

### A. Anomaly Model

We aim at understanding how the device reacts to failures, and detect the performance anomalies that these events generate. Therefore, we contacted the stakeholder to discuss about the way the target ARANCINO device was made, potential vulnerabilities, existence of bottlenecks and relevant software or communication channels. Then, we scanned the literature to seek for fault models that apply to a Linux-based embedded system / IoT device [11], [12], [13]. There is an overall agreement about the likelihood of one of the following events happening in a Linux-based OS.

- Resource consumption: either CPU, primary and secondary memory may be filled / exhausted by malicious or malfunctioning software
- Deadlock: critical sections are heavily used in any multi-threading context. A shallow management of locks or semaphores may end up generating deadlocks and make the regular execution flow deviate from expectations.
- Unexpected usage of network, in both directions.

On top of that, we consider that ARANCINO devices heavily rely on the Redis [4] database: therefore we also consider erroneous usages of the Redis database, which we simulate as subsequent reads / write operations. Lastly, we disturb the regular usage of key processes that manage the overall device, namely the arancino and node-red Raspbian processes, and make them stuck for some time to simulate their potential malfunction.

This leads to a total of 8 different groups of faults (CPU usage, RAM usage, Disk Usage, Deadlock, Redis read, Redis write, Stuck arancino, Stuck node-red) that we will inject into our device, monitoring its behaviour in the process.

### B. A Lightweight Monitor for Linux-based Devices

The successive step is to equip the device with a monitor that has the following requirements: i) lightweight, ii) customizable regarding sampling interval and the system indicators to observe, iii) able to instrument different layers and components of the target system, iv) compatible with the *Raspbian 9 Stretch* system, the OS running on the ARANCINO devices. This means that the tool has to be either written in C/C++ (gcc 7.x), Python <= 3.5.3, or Java (v. 8 openJDK).

Unfortunately, we did not find anything reusable: as such, we coded our monitor ourselves, and made it publicly available through a public GitHub repository [3]. The monitor is written in Python 3.5.3 and equipped with a total of 7 probes, that can be activated at will:

- Network (32 features): reads data from the system file `/proc/net/dev`

- Chip temperature (1 feature): reads data from the system file `/sys/class/thermal/thermal_zone0/temp`
- Virtual Memory (116 features): reads data from the system file `/proc/vmstat`
- Memory Info (38 features): : reads data from the system file `/proc/meminfo`
- IO Stats (6 features): uses the `iostat` Linux package and parses its textual output
- Python Indicators (55 features): uses the `psutil` functions `cpu_times`, `cpu_stats`, `getloadavg`, `swap_memory`, `virtual_memory`, `disk_usage`, `disk_io_counters`, `net_io_counters`.
- Redis DB (25 features): accesses to Redis performance indicators through the `redis-py` Python wrapper

This monitor has only minimal dependencies and thus can be installed without requiring to download additional libraries. For further information, please refer to the documentation available at [3].

### C. Experimental Campaigns

We installed the monitor above into our target device and set the monitor to run, logging performance indicators once a second, while the ARANCINO was performing its usual tasks. Additionally, we prepared a script that injects the 8 faults from Section II.A. This injection is performed as follows: i) it activates with a given probability, and randomly chooses one of the 8 faults, ii) it lasts for a given amount of time (5 seconds in our setup), then the device is left alone for a cooldown period that makes it recover from the previous injection (5 seconds in our setup), iii) the timestamps of activation and de-activation of the error are then logged into a dedicated file.

This provides us with an experimental testbed that we can activate at will and use to retrieve a virtually infinite amount of data, which is either corresponding to normal data or to the behaviour of the ARANCINO, while a specific error was injected. In other words, we collect a labelled dataset composed of:

- the timestamp;
- a total of 276 system indicators, some of them remaining constant throughout the duration of the experiments and thus to be discarded at a later stage;
- the label, a categorical field with 9 possible values i.e., normal or any of the 8 faults.

### D. Anomaly Detection

A labelled dataset enables the usage of any supervised ML algorithm for detecting performance anomalies. This opens the ground for a plethora of different experiments and comparisons between detection performance of a multitude of algorithms. However, literature tells that the de-facto standard for processing tabular datasets is to use tree-based ML algorithms such as Decision Trees, Random Forests, (eXtreme) Gradient Boosting, Extra Trees, and others. Those algorithms typically outperform neural networks, even those that are being re-shaped to explicitly classify tabular data [15].

## III. WHAT'S NOW AND WHAT'S NEXT

We are currently training different ML algorithms to learn how to detect performance anomalies in the ARANCINO

device. This process is being carried out carefully to avoid common pitfalls [1] and using appropriate metrics for evaluation [2]. After the learning phase, we will deploy the learned models to the ARANCINO device, and quantify:

- The false alarms they raise, and the fraction (coverage) of the errors that are correctly detected
- Their response time, space and memory occupation, which are a typical concern when dealing with resource-constrained devices.

This will allow to choose the preferred ML algorithm for anomaly detection, complete the deploy and starting to plan how to take advantage of the alerts delivered by the anomaly detector to take automatic countermeasures and mitigate the occurrence of potential threats to the device. All software and dataset will be released publicly.

### ACKNOWLEDGEMENTS

This work was partially supported by project SERICS (PE0000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU. We thank eng. Francesco Alessi from SmartMe.io for his support in preparing this paper.

### REFERENCES

- [1] Arp, D., et al. (2022). Dos and don'ts of machine learning in computer security. In 31st USENIX Security Symposium (USENIX Security 22) (pp. 3971-3988).
- [2] Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient over F1 score and accuracy in binary classification evaluation. *BMC genomics*, 21, 1-13.
- [3] Lightweight Linux Monitor GitHub (online), <https://github.com/tommyipoz/arancino-monitor>
- [4] Carlson, J. (2013). *Redis in action*. Simon and Schuster.
- [5] Murshed, M. S., et al. (2021). Machine learning at the network edge: A survey. *ACM Computing Surveys (CSUR)*, 54(8), 1-37
- [6] Zhu, G., et. al. (2020). Toward an intelligent edge: Wireless communication meets machine learning. *IEEE communications magazine*, 58(1), 19-25.
- [7] Merenda M, Porcaro C, Iero D. (2020). Edge machine learning for ai-enabled iot devices: A review. *Sensors*, 20(9), 2533
- [8] Koopman, P., Sung, J., Dingman, C., Siewiorek, D., & Marz, T. (1997). Comparing operating systems using robustness benchmarks. In *Proceedings of SRDS'97*, pp. 72-79. IEEE.
- [9] Zoppi, T., Ceccarelli, A., Bondavalli, A. (2019). MADneSs: A multi-layer anomaly detection framework for complex dynamic systems. *IEEE Transactions on Dependable and Secure computing*, 18(2), 796-809.
- [10] Chou, A., et al D. (2001, October). An empirical study of operating systems errors. *Proc of the 18<sup>th</sup> ACM Symp. on OS principles* (pp. 73-88).
- [11] Shwartz-Ziv, R., & Armon, A. (2022). Tabular data: Deep learning is not all you need. *Information Fusion*, 81, 84-90.
- [12] Gorishniy, Y., et al. (2021). Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34, 18932-18943.
- [13] Giacobbe, M., Alessi, F., Zaia, A., & Puliafito, A. (2020). Arancino.cc™: an open hardware platform for urban regeneration. *International Journal of Simulation and Process Modelling*, 15(4), 343-357.