Low-level Anomaly Detection in Embedded Systems Using Machine Learning

1st Carlo Vitucci*
Technology Management
Ericsson AB
Stockholm, Sweden
carlo.vitucci@ericsson.com
*Corresponding author

2nd Daniel Sundmark
Computer Science and Software Enigineering
Mälardalen University
Västerås, Sweden
daniel.sundmark@mdu.se

3rd Marcus Jägemar
Sys Compute Dimensioning
Ericsson AB
Stockholm, Sweden
marcus.jagemar@ericsson.com

4th Thomas Nolte
Division of Networked and Embedded System
Mälardalen University
Västerås, Sweden
thomas.nolte@mdu.se

***Abstract*—Let us consider an embedded system as a specific combination of hardware and software that is capable of consistently providing a certain service. Depending on the boundary conditions of the system, such as the working environment and the number of users served, we can say that the statistical distribution of resource usage is a characterization of the embedded system itself and its footprint. The consequence of this distinguishable footprint for embedded systems is that it becomes possible to use the statistical deviation of the resource usage distribution to identify anomalies. In this paper, we will analyze which Performance Metric Unit counters (e.g., CPU usage, memory usage) and resource profiles (e.g., system logs, performance metrics) are most characteristic for detecting a low-level anomaly: an alteration of the firmware working cycle or the propagation of a hardware error in the system. We will do this by using baseband products for radio access networks. We will demonstrate that using a machine learning model makes it possible to distinguish both the firmware cycle alteration and the hardware error reporting with an accuracy of more than 99% on unseen and new dataset.**

***Index Terms*—Anomaly Detection, Machine Learning, Embedded System, Fault Detection**

I. INTRODUCTION

An embedded system is a specific combination of hardware and software designed to perform a certain function or provide a certain service [1]. Although updating the software running on the system is always possible, an embedded system operates under fixed principles, consistently reacting to external stimuli based on predetermined design rules. This consistency allows the system to exhibit deterministic behaviour (hard real-time) or quasi-deterministic behaviour (soft real-time). As a result, an embedded system displays an execution profile that, based on input and working conditions, shows a statistically constant hardware resource usage distribution over time. Akin to what might be considered the system's "fingerprint." In the Radio Access Network (RAN) domain, the Baseband Unit (BBU) is a crucial embedded system dedicated to performing encoding, decoding, modulation, demodulation, and other digital operations required to handle signals to and from the radio units (Remote Radio Heads, or RRH) for wireless transmission [2]. The most common mechanism for signalling between the central processor and devices in the system is the interrupt. The propagation of interrupts follows a three-level process (see Figure 1): a hardware device either signals information or detects a possible malfunction and generates an interrupt — a trigger asynchronous to the software execution — which is sent to the CPU. The firmware, microcode hosted into BIOS, UEFI or low-level software, processes the interrupt, translating it into an exception recognized by the operating system. The OS then suspends the current software execution, calls a dedicated routine to interpret the specific hardware trigger, and, based on the result, determines which task to execute next [3]. The impact of interrupts on software execution becomes clear from this description: to process devices information, the system undergoes a "suspension." If the device is conveying useful data, this suspension becomes part of the system's execution profile, ensuring the service's delivery. However, if the interrupt signals an error, the suspension reflects the overhead of error handling rather than contributing to the

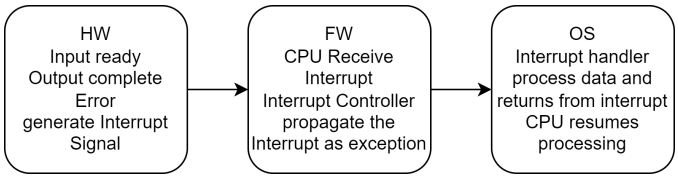


Fig. 1: Interrupt-driven path

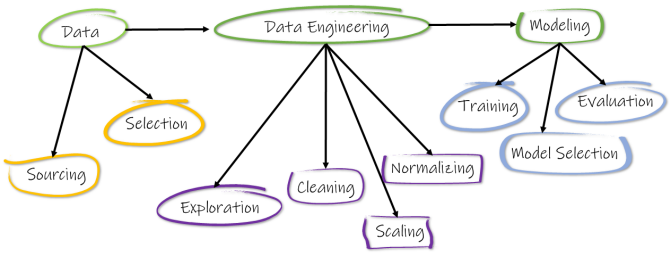


Fig. 2: Research Method

service, thus temporarily limiting system availability. In other words, error management introduces statistical deviations in the embedded system's fingerprint. In embedded systems, memory devices can report both Corrected Errors (CE), when hardware knows how to fix the issue, and UnCorrected Errors (UCE), when hardware cannot resolve the fault using ECC algorithm [4], [5].

This paper examines the impact of CE propagation on an embedded system's fingerprint. It applies machine learning techniques to perform the inverse process, detecting CE events solely through statistical variations in the system's fingerprint. We will validate this approach through two use cases. First, we use error injection to induce a functional anomaly in the system and demonstrate how our method identifies firmware-level anomalies. Second, we use a board with a corrupted memory device that periodically generates CEs for a specific address and demonstrate that our approach can also identify anomalies caused by error propagation within the operating system.

II. RESEARCH METHOD

Research Question: *Can machine learning algorithms detect anomalies in embedded systems by analysing statistical deviations in resource usage?*

The paper is a quantitative empirical study [6] that aims to examine the relationships between anomalies and resource usage using a machine learning approach. The research uses an industrial baseband board system hosting a Linux distribution based on the 6.6.23 kernel version to evaluate the anomaly detection algorithms. We will use only a limited amount of operating system features and files:

- Linux features [7]
  - **Top**: the program provides a dynamic real-time view of a running system. We use Top for a system summary information recall.
  - **free**: the program provides the total amount of free and used physical and swap memory in the system, as well as the buffers and caches used by the kernel.
- Linux files [7]
  - **/proc/slabinfo**: the file collects statistics on objects used by the kernel.
  - **/proc/stat**: the file collects kernel statistics.
  - **/proc/interrupts** the file collects the number of interrupts per CPU and IO device.

The data refer exclusively to industrial baseband boards, and this paper uses them in respect of a confidential agreement. The baseband unit runs in stand-alone, without actual traffic and Radio Access Technology (RAT) applications. The program "stress" simulates load and resource overload conditions. "stress" is a simple yet powerful tool designed to impose a configurable amount of CPU, memory, I/O, or disk stress on a Linux system. By simulating heavy workloads, allows administrators to observe how the system responds under pressure. The research method has three phases (see Figure 2):

- *Data Handling*: Understand what, how and when monitoring data.
- *Data Engineering*: Data inspection and cleaning, dataset scaling, preparing data for the model.
- *ML Modeling*: compare model's performance using training and testing dataset and verify the model with unseen data.

III. RELATED WORKS

The problem of detecting an operating anomaly for an embedded system has recently captured research attention, especially with the increasing data availability. The potential for further research in this area is vast, especially considering the increasing data availability. Employing algorithms and machine learning techniques is the most natural and logical approach for automatically analysing vast amounts of information that would otherwise be impossible to process. When analysing anomaly detection in embedded systems, the research unanimously considers the system's tendency to have a statistically traceable behaviour as an advantage. Restrepo-calle et al. [8], for example, used sampling with and without error injection and statistical variation to classify the system impacts. The study prefers relative sampling for the meters, that is, sampling the gap in the counters from the last sample. Unfortunately, the work focuses exclusively on the statistical distribution of the error and does not apply the same concept to the characterization of the system fingerprint. Also Tahmasebi et al. [9] used embedded system profiles to identify anomaly using error and error behaviour models. Another example is the work of Yoon et al. [10]. This study presents a new mechanism for identifying abnormal behaviour at the system level using the predictable nature of embedded applications in real-time. It is interesting work, but the result is limited to the detection of software anomalies without considering the validation in case of hardware anomalies. The recognition of the statistical fingerprint of embedded systems, on the other hand, is the starting point of Ezeme et al. [11]. The interesting aspect of their work is that they have recognized a deep relationship between kernel and user space with the Linux operating system. Based on this consideration, they sampled the statistical variation of the kernel to characterize the entire system. They identified anomalies through a new HAbAD model (Hierarchical Attention-based Anomaly Detection) based on Long Short-Term Memory neural networks (LSTM) [12] and with attention mechanisms [13]. Ott et al. [14] preferred to use application-related samples instead.

They use perf [15] counters to characterize the application under analysis and two methods for detecting anomalies: the Hidden Markov Model (HMM) [16] and Long Short Term Memory (LSTM) neural networks, which are trained on the data of the program's performance counters to enable classification of the program. Work that has a specific impact on the system, although quantified at less than 5%, focuses exclusively on the software anomaly. It must be clear which kernel objects are most important for system characterization or what sampling dynamics they have used. As for the algorithms used to detect anomalies in an embedded system, one of the most common methods is the K-means for clustering, which is associated with some metrics for distance determination. For example, Abbasi et al. [17] used temperature sampling and processor energy consumption as indicators of the system fingerprint. The statistical correlation of sample distribution is also a recurring algorithm. Abbas et al. [18] used it to detect anomalies in the energy consumption statistics.

IV. FIRMWARE-LEVEL ANOMALY DETECTION

The scope of this use case is to create a disturbance on the firmware layer and verify if it is possible to classify the anomaly caused by a firmware layer "intrusion" using a machine learning algorithm. We use an Error injection mechanism to add activities to the firmware layer. Using error injection means acting on a command register read by low-level software and translated into a Memory Reference Code (MRC) processor microcode action. The MRC will then generate an interrupt, following the same Signal flow (see Figure 1) as an actual Corrected Error (CE) detected by the memory controller device.

TABLE I: List of Attributes for Firmware-layer Anomaly Detection (jiffy is the Linux kernel system clock time unit. 10ms in our system)

Name	Source	Meaning
CPU		
cpu_load	Top command	CPU Load, average
CPU_sys	/proc/stat file	per core "jiffies" time in kernel execution
CPU_iow	/proc/stat file	"jiffies" time waiting for I/O completion
CPU_sof	/proc/stat file	the amount of "jiffies" servicing softirq
CTX	/proc/stat file	total number of context switches across all CPUs
MEMORY		
mem_u	free Command	Memory used in MB
mem_c	free Command	Memory available per process memory in MB
KERNEL OBJECTS		
kmalloc_32	/proc/slabinfo	Memory buffer for kernel 32bytes
kmalloc_64	/proc/slabinfo	Memory buffer for kernel 64bytes
signal_cache	/proc/slabinfo	Memory used for kernel signalling

A. Data Sourcing

Since higher data quality is mandatory for a high quality of the wanted model, we start to verify the correlation between collected counters and the CE event. *Source* in Table I shows the preliminary Linux features and files selection done. The CE correlation analysis reduced the number of attributes significant for the dataset. The sampling interval is set to 1 second, a frequency that ensures the data collected is meaningful: instead of recording the absolute values of the attributes, the dataset will reflect the differences from the most recent reading. Due to the testing conditions, a stand-alone board running a resource-stressing program, the network object counters are not significant for the fingerprint. Similarly, the constant environment temperature condition in the lab environment renders the temperature sensors insignificant for the fingerprint. The data collection is done using a bash script. Table I shows the attributes for the dataset after correlation analysis. For training the model in supervision mode, the dataset contains CE attribute that could be:

- CE = 0 if there was no CE event in the latest second
- CE = x if there were x CE events in the latest second

and it is collected using /proc/interrupts file, because the number of CE Signalling to OS is collected in THR counter (Threshold counter). The bash file is executing at a pace of 1 second. Every three sampling, an error injection is generating a CE.

B. Data Engineering

To ensure high data quality, we collected a statistically significant number of samples for each attribute to have a representative distribution. This mechanism covered all the attributes of the whole process. However, in this section, we report the results of the attributes we used to model machine learning. We filter the dataset to have only numeric values for the attributes and remove samples with CE not equal to 0 or 1, if any. The statistics population has 180713 samples with CE = 1. The data are balanced, by means of random selection of majority class population (CE = 0) that ends up in a 180713 + 180713 samples for the whole dataset.

Figure 3 shows the Correlation between the attributes. The system fingerprint as a function of CE shows helpful statistics distribution variation for CPU Kernel usage, average CPU load, context switch and memory for signalling. For ML modelling, we also considered a dataset transformation by

TABLE II: Data Engineering summary

Description	Value
Original Data Shape	361426
Transformed Data Shape	346969
Transformed Train Set Shape	274683 (80%)
Transformed Test Set Shape	72286 (20%)
Numeric Features	10
Outliers Threshold	0.05
Normalize method	zscore
Random seed	223

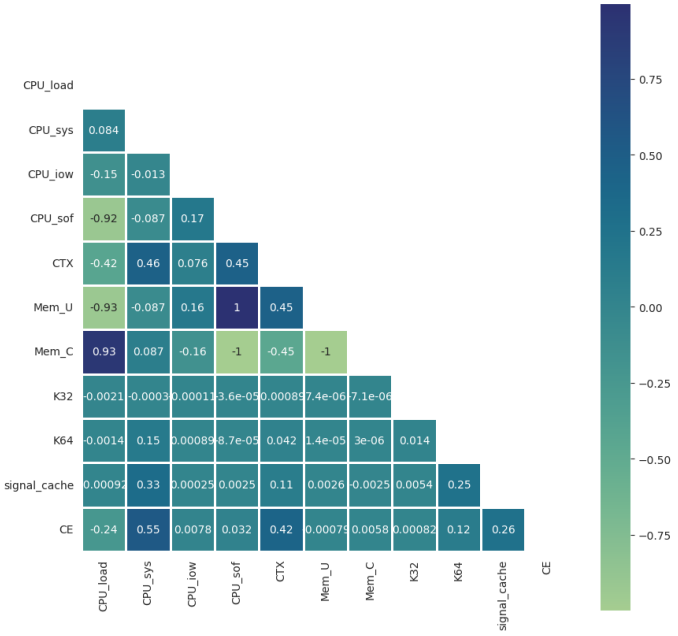


Fig. 3: Attributes correlation index

normalizing the attribute values using the zscore method [19], [20], since it will improve the accuracy of the model [21], [22]. We also remove 5% of the outliers for all attributes. Table II shows the summary of the Data Engineering phase. The setting of the random seed allows the ML model design reproducibility.

C. ML modeling

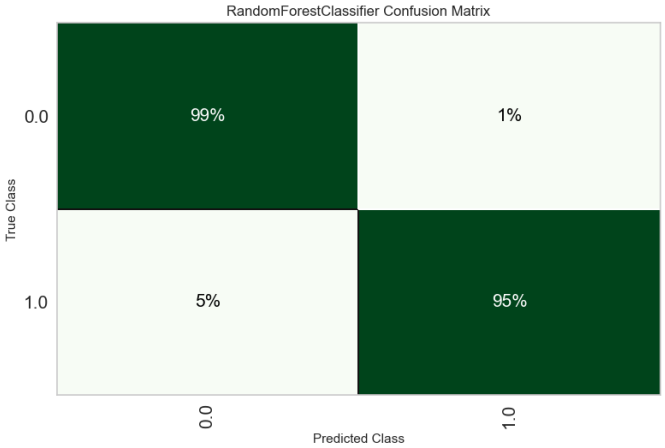
For comparing machine learning models and validating the chosen model, we used the pycaret 3.0 [23] library, an open-source, low-code machine learning library in Python that automates machine learning workflows.

1) ML model Analysis:: many researchers use clustering algorithms to determine anomalies. Our approach is to find a supervised classification mechanism. The objective of the model is not just to identify a statistical deviation, but to distinguish the anomaly's cause. In this section, we focus on the anomaly due to an alteration of the system low-level software (low-level - anomaly).

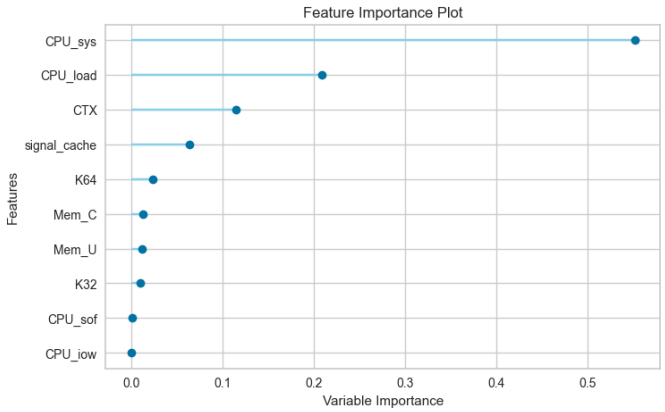
Our comparison between mathematical models of supervised classification shows that the Random Forest model yields the best result with its thorough approach (see Figure 4).

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.9777	0.9964	0.9812	0.9743	0.9778	0.9554	0.9554	15.4140
et	Extra Trees Classifier	0.9769	0.9970	0.9794	0.9746	0.9770	0.9539	0.9539	11.8290
catboost	CatBoost Classifier	0.9736	0.9968	0.9761	0.9713	0.9737	0.9473	0.9473	27.0900
lightgbm	Light Gradient Boosting Machine	0.9727	0.9966	0.9749	0.9706	0.9728	0.9454	0.9454	6.1780
mlp	MLP Classifier	0.9680	0.9948	0.9683	0.9677	0.9680	0.9360	0.9360	38.5600
gbc	Gradient Boosting Classifier	0.9674	0.9942	0.9668	0.9679	0.9674	0.9348	0.9348	15.2820
ada	Ada Boost Classifier	0.9664	0.9947	0.9689	0.9641	0.9665	0.9329	0.9329	7.9080

Fig. 4: ML models' quality parameters comparison



(a) Confusion Matrix



(b) Feature Importance

Fig. 5: Random forest classifier model results

Supervised classification with a Random Forest model is a

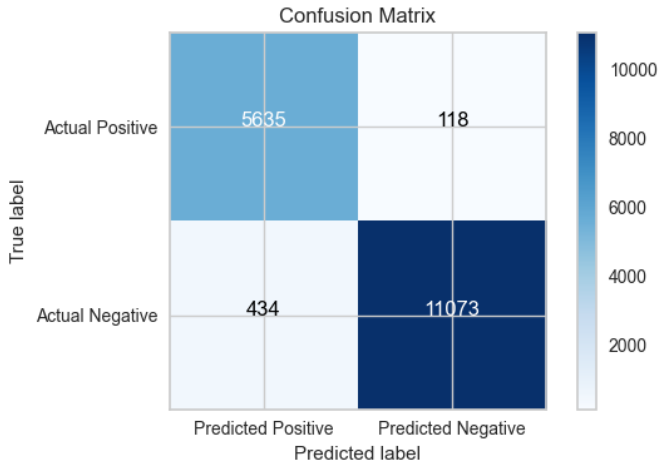


Fig. 6: ML models result using a new dataset

type of machine learning in which the objective is to predict the class (the supervised attribute) based on the other attributes (features) in the dataset. The precision (accuracy) of 99% indicates that the model correctly predicts the outcome in about 99% of the cases. Mathematically, the accuracy is:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Samples}}$$

In our case, an accuracy of 99% means that the model makes errors only in 1% of the predictions. This value reflects the ability of the model to capture the relationships between the attributes and the target class, proving our initial hypothesis that it is possible to predict an anomaly in terms of statistical variation of the system fingerprint.

From a physical point of view, the result (see Figure 5.(b)) reflects that the main attributes such as *CPU_sys*, *CPU_load*, *CTX*, *signal_cache* and *K64* are the ones that the model considers most relevant. In practice, these variables mainly determine the system's behaviour, suggesting that the supervised anomaly exclusively impacts these few system metrics. Conversely, we could say that the anomaly has minimal but still meaningful impacts on the load (*CPU_load)*, on the system performance (*CPU_sys)*, and on the kernel objects (system memory (*K64*), signalling memory (*signal_cache*) and context switch (*CTX*)).

The confusion matrix (see Figure 5.(a)) shows how the model's predictions compare to the actual values. The model missed 5% of the time when it should have predicted positively and misclassified 1% of the samples as positive when they were not. An accuracy of 99% is an excellent result and suggests that the Random Forest model is accurate. However, the 5% false negatives indicate a margin of risk in missing some anomalies, while the 1% false positives are less worrying but may indicate unnecessary alarms. The most critical attributes (such as *CPU_sys* and *CPU_load*) are the ones that play the most significant role in the correct classification, and the model relies mainly on them to distinguish the classes. Eventually, we verify the quality of the model using a new dataset batch. Attributes data are collected and clean using the same methods as described above. Figure 6 shows the result: accuracy is still higher than 99%.

V. HW FAULT AS ANOMALY DETECTION

The purpose of this second use case is to measure the cost of propagating an error generated by hardware to the operating system. It is, therefore, essential to avoid intrusive techniques, such as error injection, to avoid falling into precisely the same mode as the first use case. In other words, this test requires a baseband that contains an actual hardware fault. The operating system should not see the expectation as a fatal condition. Otherwise, it invokes a restart. For this purpose, we used a memory device affected by a single-bit flip. We have activated a policy of background patrol scrubbing [24] in the board firmware so that, periodically, the memory controller performs ECC checks over the whole memory. The result is a board that produces 5-7 CE interrupts in one hour. Of course, the CE = 1 class will be highly unbalanced, and we must face this problem during data engineering.

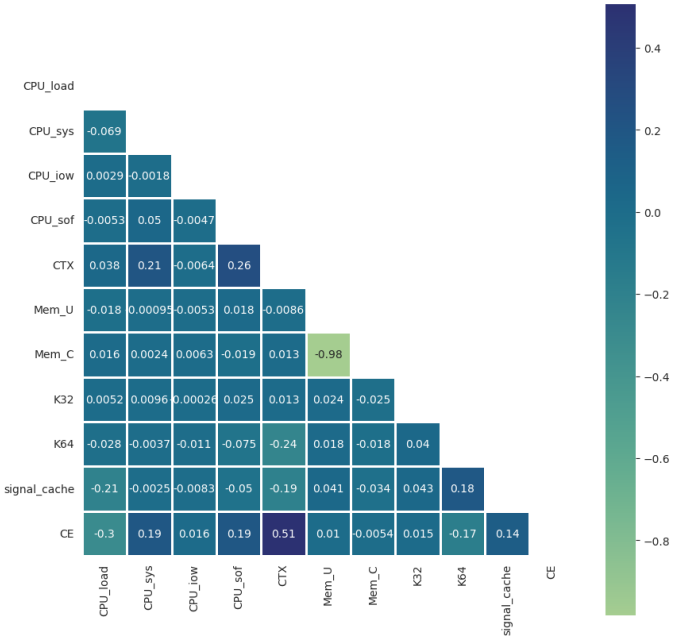


Fig. 7: Attributes correlation index, after balancing method

A. Data Sourcing

The data sourcing with this use case is following the method as described in Section IV, but with two essential differences:

1) The sampling is done using a faulty board, where the memory device contains a single-bit flip error.
2) The sampling pace is 1 second, and we must under no circumstances use error injection.

B. Data Engineering

For the statistical analysis of resource usage distribution, we collected more than 738000 samples in different data batches, of which only 4200 reported a fault condition from the memory controller — something a bit more than 0.5 % of the total. The impressive imbalance for the minority class needs action or the machine learning model will be of low quality, suffering from the problem of bias towards the majority class and misleading accuracy. We know that the SMOTE balancing technique [25] alters the statistical distribution of attribute values since the mechanism generates completely new synthetic samples [26]. Instead, we preferred the random oversampling technique that does not alter the statistical distribution of the attributes [27]. We included multiple data batches up to fifty times until generating 95193 samples of class CE = 1. Therefore, considering CE = 1 as the minority class, we selected the same number of samples for the majority class in a completely random way, obtaining the balance of the classes. With the balanced dataset, we applied the same data engineering technique as described in Section IV.

Figure 7 shows the correlation between the attributes after balancing method. The result has similar correlation values

TABLE III: Data Engineering summary, Faulty board

Description	Value
Original Data Shape	190386
Transformed Data Shape	182799
Transformed Train Set Shape	144721 (80%)
Transformed Test Set Shape	38078 (20%)
Numeric Features	7
Outliers Threshold	0.05
Normalize method	zscore
Random seed	223

of the original dataset, proving the quality of the balancing method. The interesting outcome from the correlation graph is that even the fault board use case has the CE fingerprint as a function of CPU Kernel usage, average CPU load, context switch and memory for signalling. Table III shows the summary of the Data Engineering phase.

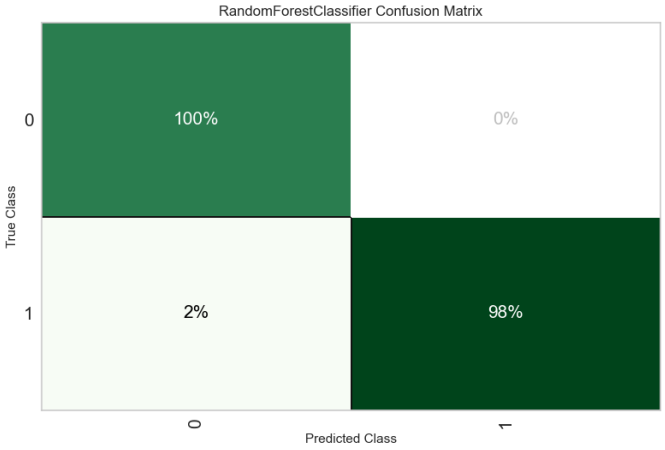
C. ML modelling

Section IV-C describes the ML model accuracy, comparison methodology and the reason behind the decision to use classification and not clustering mechanism for anomaly detection. Those concepts also remain valid in the actual faulty board use case.

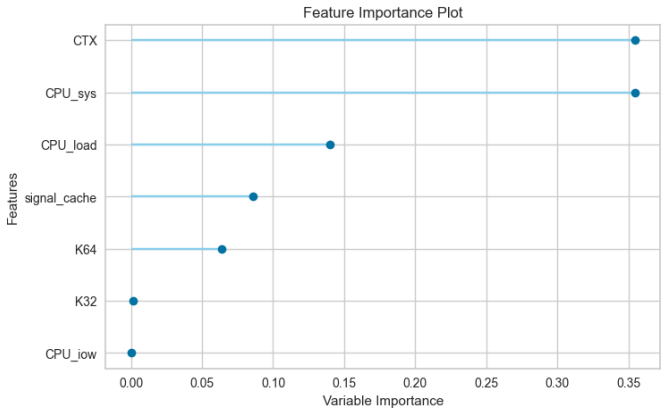
1) ML model Analysis:: the random forest classifier model performs better than the others, even in the faulty board case (see Fig. 8). The precision (accuracy) of 99% indicates that the model correctly predicts the outcome in about 99% of the cases. Fig. 9.(a) shows that the model is working efficiently with the false positive and generates more false negative. Fig. 9.(c) shows the measure of quality when we make the model work on a new dataset, that is, a new batch of data distinct from those used for training and testing the model. The verification shows that the false positive rate is about 11%, while the false negative rate is well below 1%. Fig. 9.(b) shows the most importance features. In a physical point of view, they match those used by the model in Section IV-C.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.9958	0.9987	0.9943	0.9973	0.9958	0.9916	0.9917	5.6880
et	Extra Trees Classifier	0.9956	0.9987	0.9936	0.9976	0.9956	0.9912	0.9912	4.6090
catboost	CatBoost Classifier	0.9920	0.9983	0.9912	0.9928	0.9920	0.9840	0.9840	16.8620
knn	K Neighbors Classifier	0.9912	0.9953	0.9921	0.9903	0.9912	0.9824	0.9824	5.0490
dt	Decision Tree Classifier	0.9891	0.9891	0.9878	0.9903	0.9890	0.9781	0.9782	2.5540
lightgbm	Light Gradient Boosting Machine	0.9760	0.9973	0.9626	0.9890	0.9756	0.9519	0.9522	3.0330
gbc	Gradient Boosting Classifier	0.9534	0.9823	0.9253	0.9803	0.9520	0.9068	0.9082	6.1580
ada	Ada Boost Classifier	0.9464	0.9792	0.9208	0.9705	0.9450	0.8927	0.8939	3.7590
svm	SVM - Linear Kernel	0.8965	0.0000	0.8332	0.9540	0.8895	0.7929	0.7994	2.7690
lr	Logistic Regression	0.8954	0.9417	0.8476	0.9372	0.8902	0.7908	0.7945	3.3670
ridge	Ridge Classifier	0.8538	0.0000	0.7140	0.9912	0.8297	0.7076	0.7372	2.4760
lda	Linear Discriminant Analysis	0.8538	0.9196	0.7140	0.9912	0.8297	0.7076	0.7372	2.6670
nb	Naive Bayes	0.8504	0.9694	0.7028	0.9974	0.8245	0.7009	0.7336	2.4470
qda	Quadratic Discriminant Analysis	0.8055	0.9696	0.6120	0.9983	0.7588	0.6110	0.6626	2.6050
dummy	Dummy Classifier	0.5000	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	2.5390

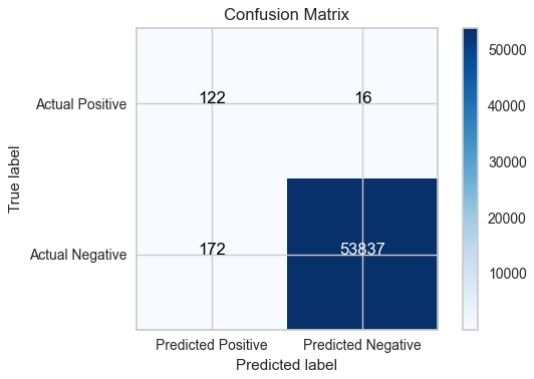
Fig. 8: ML models quality parameters comparison, faulty board



(a) Confusion Matrix



(b) Feature Importance



(c) Quality parameter with new dataset

Fig. 9: Random forest classifier model results, faulty board

Analysing the different weights of the features in the two cases allows a better understanding of what happens in the system. In the case of low-level software anomaly, the kernel's load is necessarily higher (average value is 168.13 jiffies). Instead, in the case of faulty boards, there is only the last part of the interrupt flow (kernel load average value for CE=1 event

is 54.159 jiffies), the signalling to the operating system. So *CPU_sys* loses weight, and *CTX* raises weight since the need to handle an exception will inevitably cause an increase in the number of context switches. Eventually, Fig. 9.(c) shows the high model performance with a new dataset batch.

VI. CONCLUSIONS AND FUTURE WORKS

The statistically constant behaviour over time for the embedded system translates into a resource usage whose statistical distribution becomes the system's fingerprint. We considered two use cases: the ability to detect a firmware anomaly and the ability to intercept a hardware error condition in the system memory. We used a baseband product for the radio access network. We tried to find a supervised machine learning model to identify the statistical variation in resource usage due to low-level anomaly (firmware or hardware). The random forest classifier is the best model for both cases, with excellent performance: 99% accuracy for the test data. As a future development of this project, it will be interesting to include metrics for actors known to have a weight in determining the system fingerprint: the number of users and the environmental conditions. The number of users and managed Ethernet traffic are clearly connected. In a separate study [28], we demonstrated the direct relationship between ambient conditions and the temperature sensors available in the hardware architecture. We can enhance the model by incorporating temperature sensors along with Ethernet traffic data—both incoming and outgoing—before proceeding to a new round of training.

ACKNOWLEDGEMENTS

The work presented in this paper is sponsored by Ericsson, Mälardalen University and the Swedish Knowledge Foundation (KKS), via the industrial PhD School ARRAY.

REFERENCES

[1] S. F. Barrett, *Embedded Systems Design*, 2024, vol. Part F1960.
[2] S. Ahmadi, *5G NR: Architecture, technology, implementation, and operation of 3GPP new radio standards*, 2019.
[3] G. G. Silberschatz, *Operating System concepts essentials*, 2018, vol. 11.
[4] R. Rooney and N. Koyle, "Micron® ddr5 sdram: New features," *Micron Technology Inc., Tech. Rep*, 2019.
[5] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.
[6] D. Escudero-Mancebo, N. Fernández-Villalobos, Óscar Martín-Llorente, and A. Martínez-Monés, "Research methods in engineering design: a synthesis of recent studies using a systematic literature review," *Research in Engineering Design*, 2023.
[7] M. Kerrisk, *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*, 1st ed. USA: No Starch Press, 2010.
[8] F. Restrepo-Calle, S. Cuenca-Asensi, M. A. Aguirre, F. R. Palomo, H. Guzmán-Miranda, and A. Martínez-Álvarez, "On the definition of real conditions for a fault injection experiment on embedded systems," in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*, 2011, pp. 497–500.
[9] K. N. Tahmasebi and D. J. Chen, "A fault injection tool for identifying faulty operations of control functions in automated driving systems," in *Lecture Notes in Networks and Systems*, vol. 484 LNNS, 2022.
[10] M. K. Yoon, S. Mohan, J. Choi, and L. Sha, "Memory heat map: Anomaly detection in real-time embedded systems using memory behavior," in *Proceedings - Design Automation Conference*, vol. 2015-July, 2015.
[11] M. O. Ezeme, Q. H. Mahmoud, and A. Azim, "Hierarchical attention-based anomaly detection model for embedded operating systems," in *Proceedings - 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2018*, 2019.
[12] A. Chandra and R. Kala, "Regularised encoder-decoder architecture for anomaly detection in ecg time signals," in *2019 IEEE Conference on Information and Communication Technology, CICT 2019*, 2019.
[13] M. H. Guo, T. X. Xu, J. J. Liu, Z. N. Liu, P. T. Jiang, T. J. Mu, S. H. Zhang, R. R. Martin, M. M. Cheng, and S. M. Hu, "Attention mechanisms in computer vision: A survey," 2022.
[14] K. Ott and R. Mahapatra, "Hardware performance counters for embedded software anomaly detection," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 528–535.
[15] A. C. de Melo, "The new linux 'perf' tools," *Linux Kongress*, 2010.
[16] S. R. Eddy, "What is a hidden markov model?" 2004.
[17] Z. Abbasi, M. Kargahi, and M. Mohaqeqi, "Anomaly detection in embedded systems using simultaneous power and temperature monitoring," in *2014 11th International ISC Conference on Information Security and Cryptology, ISCISC 2014*, 2014.
[18] M. F. B. Abbas, A. Prakash, and T. Srikanthan, "Power profile based runtime anomaly detection," in *Proceedings - 2017 TRON Symposium, TRONSHOW 2017*, vol. 2017-January, 2017.
[19] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, "An introduction statistical machine learning with applications in python," *Springer Texts in Statistics*, 2023.
[20] F. Sohil, M. U. Sohali, and J. Shabbir, "An introduction to statistical learning with applications in r," *Statistical Theory and Related Fields*, vol. 6, 2022.
[21] R. S. Kumar, A. Arulanandham, S. Arumugam, G. Dinesh, R. Thirukkumaran, and R. Subashmoorthy, "Analysis of classification and clustering techniques for ambient aqi using machine learning algorithms," in *Proceedings - 4th International Conference on Smart Systems and Inventive Technology, ICSSIT 2022*, 2022.
[22] K. Cabello-Solorzano, I. O. de Araujo, M. Peña, L. Correia, and A. J. Tallón-Ballesteros, "The impact of data normalization on the accuracy of machine learning algorithms: A comparative analysis," in *Lecture Notes in Networks and Systems*, vol. 750 LNNS, 2023.
[23] M. Ali, *PyCaret: An open source, low-code machine learning library in Python*, April 2020, pyCaret version 1.0. [Online]. Available: https://www.pycaret.org
[24] C. Vitucci, D. Sundmark, J. Danielsson, M. Jägemar, A. Larsson, and T. Nolte, "Run time memory error recovery process in networking system," in *2023 7th International Conference on System Reliability and Safety (ICSRS)*, 2023, pp. 590–597.
[25] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, 2002.
[26] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Learning from Imbalanced Data Sets*, 2018.
[27] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Handling imbalanced datasets : A review," *Science*, vol. 30, 2006.
[28] S. Rahman, M. Olausson, C. Vitucci, and I. Avgouleas, "Ambient temperature prediction for embedded systems using machine learning," in *Engineering of Computer-Based Systems*, J. Kofroň, T. Margaria, and C. Seceleanu, Eds. Cham: Springer Nature Switzerland, 2024, pp. 12–25.