Improving the Robustness of Industrial Cyber-Physical Systems through Machine Learning-Based Performance Anomaly Identification

URAZ ODYURT, University of Amsterdam, The Netherlands
ANDY D. PIMENTEL, University of Amsterdam, The Netherlands
IGNACIO GONZALEZ ALONSO, ASML Netherlands B.V., The Netherlands

We propose a versatile and fully data-centric methodology towards anomaly detection and identification in modern industrial Cyber-Physical Systems (CPS). Our motivation behind this move is the ever-growing computerisation in these systems, in the form of complex distributed computing nodes, running complex distributed software. Industrial CPS also demonstrate heavy deployment of hardware sensors, as well as an increasing role for software. We observe the insufficiency and costliness of design-time measures in prevention of anomalies. As our main contribution, our methodology is taking advantage of this data-rich environment by means of Extra-Functional Behaviour (EFB) monitoring, analytics pipelines and Artificial Intelligence (AI). Specifically, we demonstrate the use of compartmentalisation of execution timelines into distinct units, i.e., *execution phases*. We introduce the generation of representations for these phases, i.e., *behavioural signatures* and *behavioural passports*, as our way of behavioural fingerprinting. Composed using regression modelling techniques, signatures as the representation of ongoing behaviour, are compared to passports, representing reference behaviour. The comparison is done by means of goodness-of-fit scores, creating quantifiable measures of deviation between different recorded behaviours. We have used both partially synthetic and real-world traces in our experiments, depending on the use-case. We have also followed both white box and black box approaches for our use-cases, with discussions on the pros and cons of each.

The effectiveness of our data-centric methodology is demonstrated by two proofs-of-concept from the industry, to represent the two ends of the industrial CPS complexity spectrum, with one being a large semiconductor photolithography machine, while the other is an image analysis platform. Each use-case comes with its own characteristics and limitations, confirming the flexibility of our methodology and the relevance of its integral steps in the approach towards the initial analysis and data transformations. The results of anomaly classification show overall high accuracies, as high as 99% in certain set-ups. These results show the capability of our data-centric methodology, suiting the presented modern industrial CPS designs.

Additional Key Words and Phrases: Industrial cyber-physical systems, Anomaly detection, Anomaly identification, Fault tolerance, Classification

1 INTRODUCTION

The challenge of anomaly detection and prediction during the operation of Cyber-Physical Systems (CPS) is a profound one. Current industrial evolution has brought forth the emergence of CPS in industrial solutions. Today, ubiquitous deployment of CPS in production and machinery, means that our economy and lifestyle is heavily reliant on the proper functioning of these *industrial CPS* [32]. The risk carried along with anomalies in industrial CPS is an economic one and in certain critical deployments, even life-threatening.

As industrial CPS evolve, classic control subsystems are computerised more than ever. Consequently, the role of software is ever-increasing in these systems. The continuous rise in the steering role of software alongside the heterogeneous, distributed and multi-node design of industrial CPS, has created a sharp rise in their complexity. Although model-based design practices [22] have boosted design capabilities, CPS complexity is still hard to tackle. Not

Authors' addresses: Uraz Odyurt, University of Amsterdam, Informatics Institute (IvI), Amsterdam, The Netherlands, u.odyurt@uva.nl; Andy D. Pimentel, University of Amsterdam, Informatics Institute (IvI), Amsterdam, The Netherlands, a.d.pimentel@uva.nl; Ignacio Gonzalez Alonso, ASML Netherlands B.V., Veldhoven, The Netherlands, ignacio.alonso@asml.com.

every aspect of system's operation and not every corner case is covered at design time [11]. A complete exploration of the *behavioural space* has simply become too expensive. There is a need for behavioural tracking of industrial CPS during their operation, while their sheer complexity means that following a single observable is not revealing enough.

Take the example of flight software for space missions. As it is argued in [13], most of the mission issues in the recent years are related to software. Both manned and unmanned missions have shown steady exponential growth in source lines of code from 1986 to 2004. Similar trends are present for military aircraft embedded software, taking over a bigger portion of the total available functionality [13]. The software from a modern commercial airliner has 7 M lines of code.

CPS are an amalgamation of machinery, sensors, embedded computing and communication subsystems. The types of CPS incorporated in the industry, industrial CPS, have all of the above pieces tuned towards a specific task, making them *purpose-built*. Especially, a plethora of sensors are incorporated in today's industrial CPS. These sensors include both hardware and software varieties and more importantly, what they have in common is large amounts of generated data. Though it is no trivial task to manage this data and make sense of it, if done with tact, it is an almost unlimited source to reveal all the behavioural secrets of a system. As such, it is only logical to take advantage of this data-rich ecosystem and move towards data-centric solutions, which will in turn allow employment of techniques based on Artificial Intelligence (AI).

1.1 Data-centric solutions

The large amount of data generated by production machinery sensors, has implications that are twofold. On the one hand, such a characteristic results in a need towards treating these systems with data-centric methodologies. Ergo, data-centricity is a virtue that could enrich the analytical capabilities over such systems. On the other hand, the data-centric approach has to be performed in a cautious manner, as it is rather easy to end up in a slippery slope, where there is either too much data to process in a timely fashion, or there is too much data dependence. In other words, we should somewhat limit the amount of data needed to achieve results, while at the same time, the solution should not require extremely detailed data collections. As will be shown, both risks can be circumvented by considering:

- Communication-centric monitoring and modelling of the system and
- Breaking up the operations into units of execution during the runtime of an industrial CPS.

The discussions and the methodology provided in this paper are fundamentally based on the characteristic that industrial CPS are *inherently repetitive*. Having considered repetitiveness, we show how the system's execution timeline can be compartmentalised into distinct *execution phases* and how the data contained in each phase can lead to the generation of a representative construct, a *behavioural signature* and for certain reference executions, a *behavioural passport*. We also describe how Extra Functional Behaviour (EFB) and different metrics reflecting them can be captured.

The high-level view of our anomaly detection and classification workflow is depicted in Figure 1. We strive to detect and identify performance anomalies for industrial CPS in an online fashion. We also strive to act upon these detections and employ actuations on the industrial CPS at hand, with the aim of thwarting performance anomalies, or reducing their detrimental effects. The actuation step however, is outside the scope of this paper as it is part of our ongoing research.

1.2 CPS complexity spectrum

Industrial CPS come in different sizes and with different levels of complexity. Examples can be systems as complex as a semiconductor photolithography machine, or something as simple as an embedded platform, interacting with the

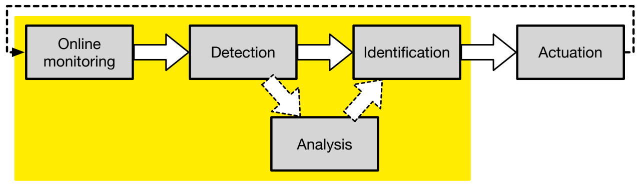


Fig. 1. High-level approach towards performance anomaly detection, identification and countering their effects, with this paper's focus highlighted in yellow

surrounding environment. With the goal of covering this vast spectrum, we have opted for cases from the extremities. Our first demonstrator is a semiconductor photolithography machine from ASML, for which the main computing node has been studied. The opposite case, is an embedded platform, used for detection of cars in images taken with the system's camera. Although both cases follow the same methodology, the exact workflows and data manipulations along the way are, to a limited extent, platform-specific.

1.3 Contribution and structure

This paper elaborates a complete view of our data-centric approach towards performance anomaly detection and identification in industrial CPS. In part, we are consolidating our previously published papers [34, 30, 33, 29, 35] into a single point of reference, as each previous individual paper was addressing only a specific part of the approach. The extra novelties complementing earlier material include: extended descriptions of the employed concepts and techniques, revised and extended diagrams, but most importantly, new experimentation with anomaly classifiers for the semiconductor photolithography machine. We are also providing extended results for our image analysis use-case, especially where classification algorithms are applied. These results reveal most effective combinations of metric and execution phase for such systems. Accordingly, we present:

- A data-centric methodology towards performance anomaly detection and classification in industrial CPS,
- Compartmentalisation of industrial CPS runtime activity into executional units, i.e., behavioural phases,
- Generation of behavioural signatures and passports as constructs reflecting the behaviour of the system during unclassified and reference executions, respectively,
- And the demonstration of our workflow with proof-of-concept set-ups for two cases from the industry.

In the following section, we briefly elaborate background concepts necessary for our data-centric approach. In Section 3, we go through our methodology and provide a relatively high-level view of the major techniques employed, while Section 4 details the implementation of our solution and data-centric pipeline. Section 5 brings forth the specifics of our experiments with proof-of-concept set-ups, while the classification results are provided in Section 6. Further discussion on using our methodology's potential and increasing its effectiveness is carried out in Section 7, followed by related literature and our conclusions in Sections 8 and 9, respectively.

2 BACKGROUND

The bulk of concepts and techniques that we use in our methodology and experiments are provided here. What follows can be seen as the tooling to take advantage of the repetitive behaviour observed in industrial CPS.

2.1 Extra-functional behaviour

Extra-Functional Behaviour (EFB) convey a computing system's behaviour and as the name suggests, EFB are not directly derived from functional aspects of the system. Examples are, execution time, different latencies, throughput, power and energy consumption, amongst others. Metrics reflecting EFB are not only dependent on functional behaviour, but also on environmental circumstances, such as the platform itself, the input to the system and operational conditions. Environmental circumstances, being important variables for CPS, necessitate the role of EFB metrics in their monitoring and analysis.

Looking at EFB from a different perspective, one can observe that they reflect the effects of environmental circumstances, the platform and the input, which may be in physical form depending on the application, all directly related to physical aspects of a CPS. As such, we can consider EFB and the information provided by EFB as an interface, binding the digital and physical realms within the operational domain of a CPS, together.

2.2 Execution phases

Execution phases are basically repeated units of execution, which are especially noticeable in industrial CPS operations. Repeated tasks can be broken down to their subtasks and higher granularity can be achieved to describe repeated units of execution. We call the smallest of these units an *atomic execution phase*. There may be (usually are) combined atomic phases that are also repetitive. Such repetitions involving two or more consecutive atomic phases are called *combo execution phases*. Figure 2 depicts these concepts under normal and anomalous execution conditions. For any particular system, analysis will reveal the best choice(s) from available atomic and combo phases.

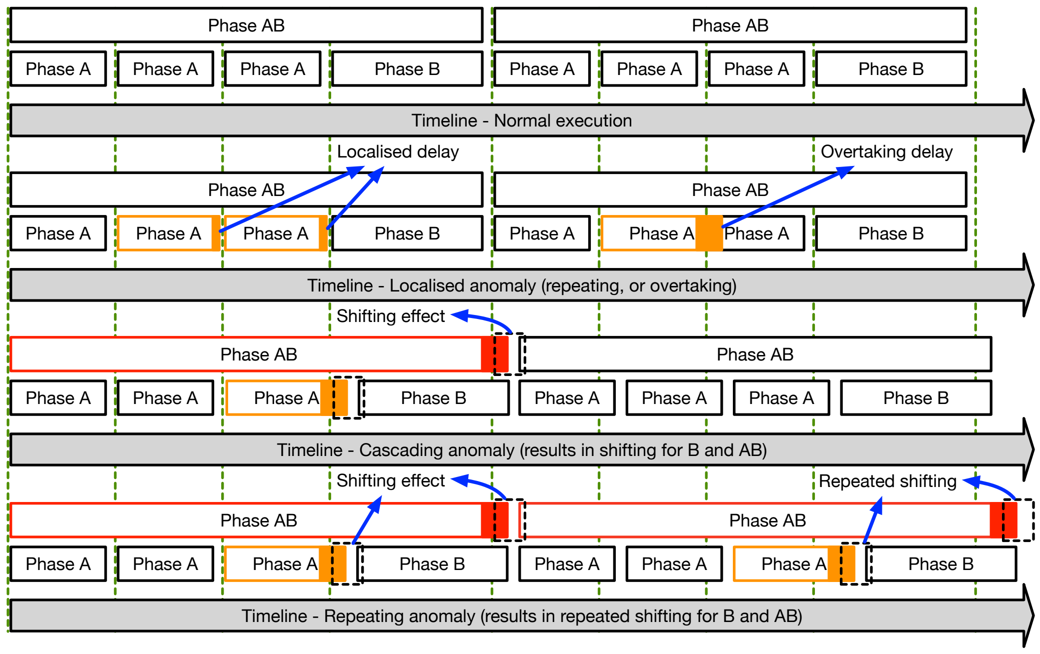

Phase AB
Phase A
Phase B
Timeline - Normal execution
Localised delay
Overtaking delay
Timeline - Localised anomaly (repeating, or overtaking)
Shifting effect
Timeline - Cascading anomaly (results in shifting for B and AB)
Repeated shifting
Timeline - Repeating anomaly (results in repeated shifting for B and AB)

Fig. 2. Showcasing the repetitive nature of application execution for industrial CPS, including atomic phases such as *Phase A* and *Phase B*, as well as the combo phase, *Phase AB*. The figure also depicts the effects of different types of anomalies related to timeliness, some localised and some with cascading effects, delaying the subsequent phases.

Formally, an arbitrary phase p is denoted as a tuple of its start, t_s, and end, t_e, times within the execution timeline, i.e., $p = (t_s, t_e)$. Each execution phase category, atomic and combo, is considered as a partially ordered set, where the generic ordering condition is $t_{e_k} < t_{s_{k+1}}$, making the ordering strict. This means that phases within one category should not overlap. As such, we can describe atomic and combo phases as

$$P_{\text{atomic}} = \{p_i : p \in T_{\text{atomic}}, i \leq n, i \in \mathbb{N}^*\} \text{ and}$$
$$P_{\text{combo}} = \{p_j : p \in T_{\text{combo}}, j \leq m, j \in \mathbb{N}^*\},$$

respectively. Here, T is the set of available phase types, n is the number of atomic phases and m is the number of combo phases. Considering the simple example from Figure 2, we have $T_{\text{atomic}} = \{A, B\}$ and $T_{\text{combo}} = \{AB\}$.

2.3 Performance anomalies

Anomalies in general can manifest themselves as unreliable or subpar performance behaviour. The manifestation might also be in other forms of unexpected and harmful system behaviour and not necessarily performance-related. The two important terms to consider here are performance anomaly and fault [33]. A *performance anomaly* is any *readily* detectable deviation in the system's performance behaviour. For instance, if a computational job takes significantly longer than it should to be fulfilled, a performance anomaly has occurred. Detecting the actual *fault* causing the performance anomaly however, requires *insight* and analysis of the internal interactions of the system. As such, a performance anomaly is the result of a fault, but at the same time, not all faults will necessarily lead to a performance anomaly. Following the notation of execution phases and considering t_s and t_e as expected start and end times vs. t'_s and t'_e as realised ones, we can expect to have the following anomalous situations:

$$t_{e_k} < t'_{e_k} < t_{s_{k+1}} \text{ and } t'_{s_{k+1}} = t_{s_{k+1}} \tag{1}$$
$$t_{e_k} < t_{s_{k+1}} \leq t'_{e_k} \text{ and } t'_{s_{k+1}} = t_{s_{k+1}} \tag{2}$$
$$t_{e_k} < t_{s_{k+1}} \leq t'_{e_k} < t'_{s_{k+1}} \tag{3}$$

Considering the relation between t'_{e_k} and $t_{s_{k+1}}$, we see that Equations (2) and (3) break the ordering condition, i.e., $t'_{e_k} \geq t_{s_{k+1}}$. These instances will be further discussed below.

Transient anomalies. A transient or localised anomaly is visible for a short period of time, or occurs only a few times. In such cases, either the fault is a short-lived one, or its impact to the overall execution is rather contained. For instance, with regards to timeliness, transient anomalies could be seen as delayed tasks in a contained part of the execution timeline, which means that the natural delays within the overall execution timeline can absorb such localised extended phase executions. We can also think of tasks without any dependency that do overlap and end up sharing available resources, as visualised in Figure 2 for the localised anomaly with independent consecutive phase As. In this context, Equations (1) and (2) point to localised and overtaking delays, respectively.

Persistent anomalies. In contrast to transient anomalies, a persistent or repeating anomaly is of the type that either will keep reoccurring, or will create cascading delays for all subsequent tasks. This could be the result of, for instance, dependencies between tasks, or excessive amount of delay. Another factor is the placement (time) of anomaly. There will be no room for compensation for an anomaly occurring towards the end of a phase. The overall cost to the execution timeline is higher for such anomalies, as depicted in Figure 2 for cascading and repeating anomalies. The cascading effect is the result of phase B being dependent on phase A, as well as each phase AB being dependent on its predecessor. Both situations result in a shifting effect, as formalised in Equation (3).

2.4 Signatures and passports

Signatures and passports [33] are data-centric representations of EFB, for they are composed using the time-series data collected during the execution timeline of an application. A signature is the representation of an execution phase, modelling the trend of EFB metrics collected during that phase. Our technique of choice for this modelling is regression. A passport is a reference signature, collected under normal and reference execution conditions, which is used in comparisons. As such, collected values for a metric of choice during an execution phase, for instance CPU time, will be transformed into cumulative data points in time, resulting in a time series collection. Using regression modelling, the closest function interpolating these data points will be generated and is used as the representation for that particular execution phase. Note that signatures are calculated per metric and per phase. For applications with multiple processes, different phases will potentially include activity data from multiple processes. This means that there will be signatures not only per metric and per phase, but also per process. Process identifiers can be used as an indicator to differentiate between signatures based on the same metric and within the same phase.

Considering our use-cases included in this paper, we have opted for both internal metrics, e.g., CPU time, and external metrics, e.g., electrical current, resulting in *software signatures* and *power signatures*, respectively.

2.5 Communication-centric monitoring, modelling and simulation

Most multi-node and multi-process, i.e., distributed, industrial CPS include a message passing subsystem, predominantly in the form of a message broker based on the *publish-subscribe* software architecture pattern. To reduce the observed behavioural surface in a complex CPS, one can limit the observation to this message passing subsystem and specifically, to the message broker process. Although the observed behaviour will be a subset of the full system behaviour, it will be a valid representation, sufficiently reflecting the trend and changes [34, 30]. In such a *communication-centric* monitoring, this is achieved by collecting traces from purposefully planted probes in the code of the target communication subsystem. Such an invasive probing provides us with enough data to support modelling and simulation efforts. Collected information consists of EFB metrics that are read and processed. In its final form, the tracing data consists of communication (read and write) and computation events. Communication events contain information such as, senders, receivers, message sizes, and timing information. Compute events contain information such as, CPU utilisation, CPU waiting times, process IDs and timing information.

We employ high-level modelling and trace-driven simulation of the CPS with the purpose of reasoning about the system behaviour. Accordingly, we have a detailed view of how different resources in the system are being used at different stages of the execution of a workload, e.g., CPU status and buffer utilisation. Moreover, the usage of high-level simulation models allows us to explore the effects of possible actuation mechanisms or countermeasures against performance anomalies in a safe environment, before applying these actuations to the real system.

High-level modelling and simulation are rather intertwined notions, as a simulation is the execution of a calibrated model. In Sections 4.1.1 and 4.1.2, we describe how trace data is used to automatically synthesise and calibrate our high-level model. The model is a simplified, but descriptive enough representation of the studied system, hence high-level. Having the knowledge of involved actor types, i.e., data producers (writers), data consumers (readers), and data brokers (communication libraries), complimented with interconnection information, i.e., topology, is sufficient to construct the high-level model. Calibrating this model for simulation is done by considering metric recordings, collected during the system's operation. The aforementioned tracing format, i.e., read, write and compute events, is especially suitable for a discrete-event simulation.

2.6 Time series data classification

As we are following a data-centric approach in our methodology, we are employing machine learning and classification algorithms [29, 35]. When it comes to classification of time series data, both deep learning algorithms, i.e., neural networks, and traditional classification algorithms, e.g., decision tree, are viable options. We are especially interested in explainable results and the ability to backtrack achievements, making connections to the original data. This level of understanding is necessary when developing a methodology. Traditional classification algorithms are much more suitable for this goal, but on the other hand, they do require *feature engineering*. Accordingly, we have experimented with Decision Tree (DT) [40], Random Forest (RF) [3], Gaussian Naïve Bayes (GaussianNB) [15], k-Nearest Neighbours (k-NN) [10], Linear Support Vector Classification (LinearSVC) [2] and Kernel Support Vector Machine (KernelSVM) [9] classifiers for our demonstrators.

3 METHODOLOGY

Considering the high-level view of our performance anomaly detection and actuation approach given in Figure 1, let us present an elaborated version here. The focus has been to take advantage of the current data-rich industrial CPS ecosystem.

3.1 Performance anomaly identification

Our methodology can be divided in two main workflows, the first of which addresses the requirement for performance anomaly detection and identification. Depicted in Figure 3, this anomaly detection and identification workflow is completely covered in this paper.

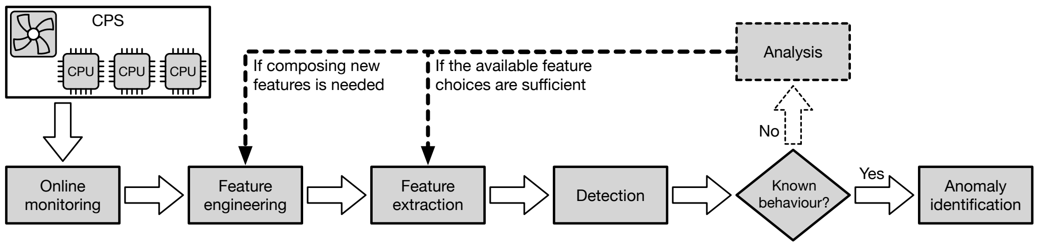


Fig. 3. High-level approach towards feature engineering, performance anomaly detection and identification

There are numerous sources of data collection in a modern industrial CPS. These include hardware-based sensors, as well as metric collection tools within the software running the system. Additional low overhead software probes and serial hardware sensors can also be introduced, as it is the case in both of our demonstrators. Considering the approach given in Figure 3, this online monitoring, happening during the operation of the system, will produce large amounts of data. The feature engineering that follows, involves different data manipulation steps. Examples are parsing, extraction of important pieces of data and especially fundamental for us, compartmentalisation. The latter is where we divide the data according to execution phases. Following feature extraction, this data set will transform into a feature set. Some features are readily taken from the data set, while others are to be calculated, e.g., through generation of signatures and passports.

Detection is the step where deviation of the behaviour at hand from the reference behaviour is considered. In practice, there is a close relation between “Detection”, “Known behaviour” and “Anomaly identification” blocks. We leave implementation details for sections ahead. Upon suspicion of new and unseen anomalous behaviour, the “Analysis” block comes into play. At this point, such an analysis is considered to be a manual effort, but we do foresee that certain analyses could be performed automatically. The goal is to update the feature set and retrain the employed classifier, allowing it for automatic detection of this newly introduced behaviour. Depending on the maturity of the feature set for this specific behaviour, there may or may not be a need to compose new features from the data set.

3.2 Behaviour improvements

Detection and identification of anomalous behaviour will be followed with another workflow aimed at countering the anomaly itself, or its effects, keeping the industrial CPS fulfilling its purpose. This workflow, given in Figure 4, is the subject of our ongoing research. However, certain pieces of the puzzle are already in place, namely, a digital twin. Not every anomaly can be countered though.

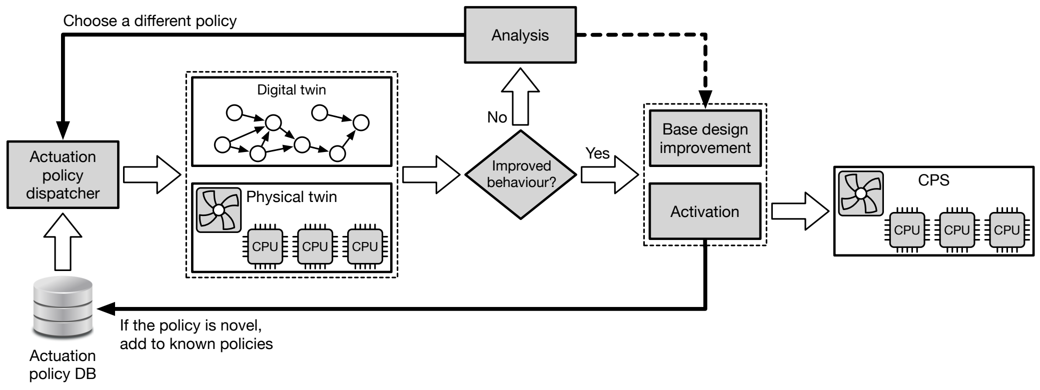


Fig. 4. High-level approach towards countering the effects of anomalies, involving actuation policy, its verification and its activation on the industrial CPS

We envision well-defined actuation policies for the anomalies that can be acted upon. The actuation policy chosen on the basis of the identified anomaly will first be applied to a twin of the system. For the most part, the anomaly detection and identification workflow from Figure 3 is also applicable to the twin. We would like to evaluate if the behaviour of the twin will be identified as normal, or if the deviation from normal is being reduced, in comparison to the anomaly itself. Upon an acceptable improvement, the actuation policy will be activated on the anomalous industrial CPS. The “Actuation policy dispatcher” may also come up with novel actuations, beyond what is previously known and available. Such policies, if resulted in improvements, will be added to the database. Whether an actuation policy results in an improvement or not, it could contribute to the base design of the system and fundamentally eliminate the anomaly from happening. For instance, when a software bug is discovered, or a corner case related to the order of calls in different processes is revealed, the base design can be improved. Although, analysis has to be performed to discover such causes behind a policy’s failure, correlate these causes with the design and develop rectifications for them. Similar to the anomaly detection and identification workflow, we expect the analysis following an unsuccessful actuation policy to be a manual one.

3.3 White box approach

A white box approach means that there is interaction with the software running the industrial CPS to collect behavioural metrics during its runtime. Whether the interaction is with the Operating System (OS), through standard/custom tooling, or through probing of the workload handling application on top of the OS, it will render the monitoring approach as white box. In such an approach, the application is analysed to discover its internal operations, e.g., I/O and computation operations. For a multi-process application, the analysis is performed on a per process basis. Note that these internal operations may or may not match the execution phases. There could very well be multiple processes that are active through different phases and different sets of operations belonging to these processes are involved in each phase.

By implementing probes collecting system-specific and application-specific EFB metrics at the beginning and end of each operation, we will be able to collect the footprint of these metrics for that specific operation. For instance, as shown in Pseudocode 1, by collecting the amount of CPU time consumed by a process at the beginning and the end of a computation operation, we will be able to calculate the amount of CPU time consumed by that operation.

Pseudocode 1: Description of software probes, with CPU time collection for a function performing computational work as an example

```
Function main():
    initialisations;
    do some work...;
    compute(input, other arguments);
    do some other work...;
Function compute(input, other arguments):
    probe and save used CPU...;
    collect_trace(processID, event, timestamp_start, CPU_used);
    computation occurs...;
    collect_trace(processID, event, timestamp_end, CPU_used);
    return
```

3.4 Black/grey box approach

The same principle as the white box approach also applies here, with the difference that no internal probing will be possible. This could be due to various reasons, such as lack of access, or strict controls over monitoring overhead. EFB metrics of choice for such cases are to be chosen from external ones. These metrics are usually collected via hardware probes in a passive fashion. The lack of interaction with cases that are being treated as black/grey box brings forth another limitation. Behaviour improvement is harder to achieve when following a black/grey box approach. We will show an example using electrical EFB metrics in the coming sections.

Hardware-based sensors are the category of choice for probing during a black/grey box approach. Especially when it comes to electrical metrics, one can use external power meters and power data loggers. Nowadays, more advanced embedded platforms have electrical metric collecting probes included in their design, reducing the need for additional hardware.

3.5 Digital and physical twins

Collecting traces under anomalous behaviour might not be a straightforward task when it comes to real production grade industrial CPS. The causes behind such a limitation can be numerous, but regardless, there will be a need for synthetically generated anomalous traces. To be able to resemble anomalous behaviour and generate synthetic anomalous traces, the role of a *digital twin* is a crucial one. This digital twin should sufficiently replicate the behaviour of its physical exemplar. Additionally, using a digital twin will dictate the application of a harmonisation procedure on the data coming from the real system. The argument here is that since both types of traces, normal and anomalous, are going to be the basis for comparisons, they have to be harmonious. The effects of synthetically introduced anomalies will not be local and there is great potential for cascading effects. As a naive example, delaying a process that should send a message to another, will certainly delay the reception at the latter, affecting its behaviour. We can achieve the conduction of such effects by using the very same digital twin as a reference platform. As such, all traces will be harmonised by passing through the digital twin. More details on this process will be given in Section 4.1.

Another important purpose of having a digital twin is its role in the actuation workflow. As seen in Figure 4, any chosen actuation policy will first be applied to a digital twin, in order to evaluate its effects on the representative twin's behaviour. Only if there was an improvement in a desirable direction, the policy will be activated in the real system. It is not a strict requirement for the twin to be a digital one. As discussed earlier, for platforms approached in a black/grey box fashion with a smaller footprint, a *physical twin* can be a fair and even more suitable option. A physical twin is especially advantageous when interacting with an industrial CPS in the form of a distributed embedded system, composed of numerous identical nodes. In such set-ups, a single physical twin can verify actuations aimed at any of the anomalous nodes.

3.6 Use-cases

Following the discussion on industrial CPS complexity spectrum from Section 1, we apply our methodology to and illustrate it for two distinct use-cases, both taken from the industry. For the case of the semiconductor photolithography machine, we follow a white box approach, in which we add probes within the software to collect EFB metrics. Thus, resulting representations are *software signatures* and *software passports*. For the case of the image analysis platform, the system has been treated as a grey box, with behaviour monitoring based on electrical metrics, which are collected via external probing. The resulting representations for this case are *power signatures* and *power passports*. Though there are subtle differences between different cases within the early data manipulation steps, generated constructs for both have been treated with the same classifiers.

4 IMPLEMENTATION

Although the implementations of behavioural signatures and their generation in both of our proof-of-concepts follow the previously elaborated methodology, differences remain. We will describe each case separately. The *software passport* workflow and the *power passport* workflow refer to the implementations of our methodology for the semiconductor photolithography machine and the image analysis platform, respectively.

4.1 Software passports workflow

As noted in Section 3, our software passport workflow has been applied to ASML's semiconductor photolithography machines, as one of this paper's real-world demonstrators. The workflow is depicted in Figure 5 and follows a white box approach. This complex industrial CPS includes communication subsystems based on the *publish-subscribe architectural pattern*. The subsystem connects application processes from different software components, which in turn run on distributed computing nodes. The depicted software pipeline has been created to extract per process, per metric and per execution phase information from the simulated traces. Regression modelling techniques are used to transform the extracted data into regression functions that are most accurately approximating the performance of processes over time.

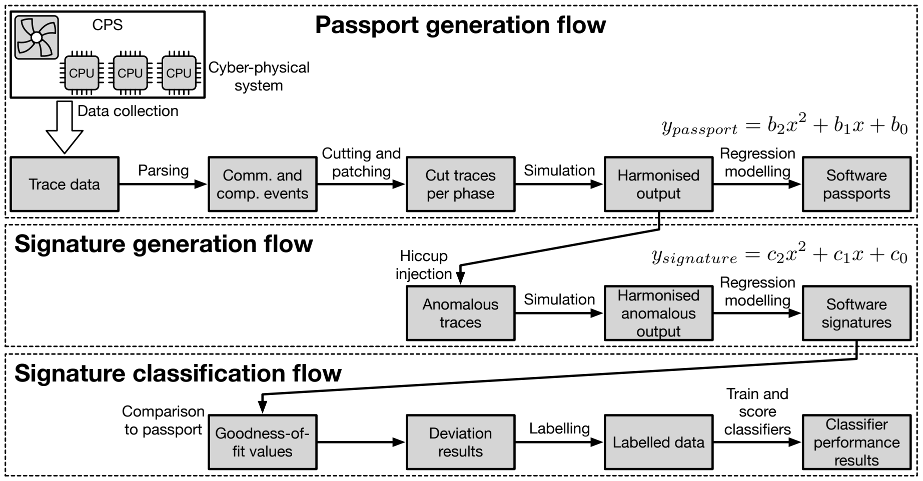


Fig. 5. Our system EFB metric collection set-up alongside the data processing pipeline for software signatures/passports, hiccup injection and classification of anomalies

For this use-case, we are working with the main computing node in the photolithography machine, which is in charge of the wafer processing and wafer exposure tasks. Thus, we have considered wafer processing operation (wafer op.) as a *combo* phase for full processing and exposure of a single wafer. We consider this operation as combo, since it can be broken down to different sub-operations, which will be more fine-grained and could be considered as atomic phases.

4.1.1 Effective sensing. We strive to capture a partial view of the system's behavioural trend, small enough for monitoring and data manipulation, but also complete enough to have a semi-accurate view of the behaviour. The sensory data required for this goal is gathered online, through invasive probing, i.e., by adding probes in the code, in a communication-centric fashion. Communication-centric monitoring allows us to limit the introduced overhead, as the software probes are implemented at the system communication module, i.e., message broker. When communication takes place, involved processes are traced indirectly from within the broker calls. The overhead is much less, compared to a fully invasive tracing of each and every software process, at the cost of a reduction in the amount of information captured. More precisely, we will miss the information on processes that are not using the communication module. The current information captured by our tracing tool includes per process data such as, event start and end timestamps, CPU utilisation, message sizes, message ids, messages sources and destinations, and memory utilisation. Following the description of software probes for a white box approach given in Section 3, Pseudocode 2 elaborates our probing for communication and computation events.

When a process starts its execution, a `trace_comp_start` call saves the id of the process, the type of event and the start timestamp. Notice that this is the initial reference point, ergo the CPU time is not recorded. The process

Pseudocode 2: Description of software probes for collection of communication and computation events at the publish-subscribe broker, within the communication module

```
Function main():
    trace_comp_start(processID, event, timestamp_start);
    do some work...;
    communicate(payload, other arguments);
    do some other work...;
    communicate(payload, other arguments);
    ...
Function communicate(payload, other arguments):
    probe and save used CPU...;
    trace_comp_end(processID, event, timestamp_end, CPU_used);
    trace_comm_start(processID, event, timestamp_start, size(payload));
    communication occurs...;
    probe and save used CPU...;
    trace_comm_end(processID, event, timestamp_end, CPU_used));
    trace_comp_start(processID, event, timestamp_start, CPU_used);
    return
```

continues its normal execution until it calls the communication library, containing the probing code, i.e., `communicate`. When entering the communication library, the `trace_comp_end` event saves the timestamp (`timestamp_end`) and the amount of resources that were used by the application until that point, while the communication event is traced using `trace_comm_start` and `trace_comm_end`. Then, collected traces are processed to calculate time spent in communication and computation related tasks. Computation events may contain not only CPU usage information, but also idle time, I/O time, etc., while communication events can also involve some CPU utilisation. The system metrics such as CPU time and memory utilisation are collected via the `getrusage` system-call.

4.1.2 Digital twin for industrial CPS. As mentioned in Section 3.5, there is a need for a digital twin when generation of anomalous traces is not straightforward. For our semiconductor photolithography machine use-case, we have to rely on a digital twin to generate anomalous traces. Accordingly, a digital twin in the form of a trace-driven simulator comes into play. The discrete-event simulation environment including real processes, simulated processes, interaction between simulated processes and the resource manager model, as well as the topology of the communication subsystem model is depicted in Figure 6. We are using the OMNEST simulation framework for this purpose [44].

Note that only processes that are using the communication module are being traced, e.g., *Real process i* will not have a contribution in the collected behavioural metrics. The outcome of tracing to be used for the modelling and simulation, contains information such as, event ID, process ID, event initiation timestamp, event end timestamp, event type, CPU utilisation, and for communication events only, message size and destination process ID. The resource manager model, i.e., hardware architecture, is modelled as a module receiving requests from different simulated processes and storing the requests in a ready queue. CPU resource management is performed using a FCFS (First Come, First Served), or a Round Robin scheduling policy. CPU time is distributed as tokens that are given to the simulated processes and returned when the processing is finished. The number of tokens depends on the number of CPU cores included in the simulated system.

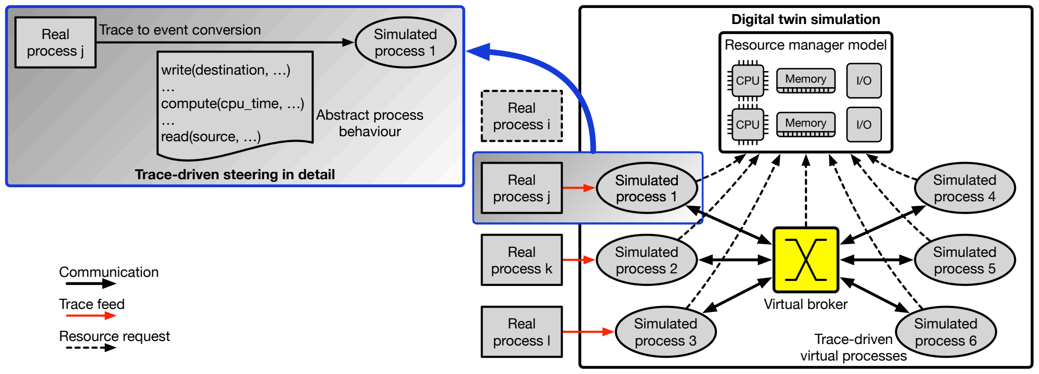


Fig. 6. A discrete-event simulation environment as the digital twin of the semiconductor photolithography machine node, considering the topology from the broker's perspective and parsed traces that are converted into *write*, *read* and *compute* events

Trace-driven virtual processes are generic simulated processes, automatically generated based on the number of executed application processes in the real system, i.e., observed processes using communication-centric monitoring. Simulated processes are trace-driven and the simulation engine executes one event at a time. Process models are automatically inferred from the monitoring traces, avoiding the need to manually derive these models, which is typically very complex, or even infeasible to do for industrial CPS. Traces are parsed and converted to three types of events, namely, *write*, *read* and *compute*. These events are consumed by simulated processes and resource requests are created depending on the content of the trace, e.g., used CPU, type of event and timestamps. For instance, a process requiring a certain number of CPU cycles to perform the operation specified in the event, will create a CPU request. This request will arrive at the ready queue of the simulated CPU and will be served according to the CPU scheduling policy implemented in the resource manager model.

4.1.3 Hiccup injection. Our fault injection implementation focuses on affecting the duration of events by modifying the end-time of events [33]. Our software allows us to completely configure the percentage of processes to be affected, the percentage of events to be affected, the type of event, i.e., communication or computation, the type of performance anomaly to be introduced, i.e., transient or persistent, as well as the overlap between the processes selected to be modified while introducing transient or persistent anomalies.

The three aforementioned types of events involve process idleness and CPU access delay. Duration of an event (t_E), which is the elapsed time from its initialisation (t_{ini}) till its end (t_{end}), is made up of CPU access delay (delay_E), CPU time (comp_E) and idleness (idle_E), such that

$$t_E = t_{\text{end}} - t_{\text{ini}} = \text{delay}_E + \text{comp}_E + \text{idle}_E.$$

We are considering synthetic increases ($\text{delay}_{\text{synth}}$) to the original duration of the event (t_E), which means that we are increasing the combined duration of CPU access delay (delay_E) and idleness (idle_E), resulting in an increased event duration ($t_{E_{\text{synth}}}$), such that

$$t_{E_{\text{synth}}} = \text{comp}_E + (\text{delay}_E + \text{idle}_E + \text{delay}_{\text{synth}}).$$

Note that CPU access delay and idleness result from different conditions and we are not able to distinguish between them using the deployed tracing mechanism. While CPU access delay is simply a wait before CPU availability, idleness could be a result of I/O waits, or functional and data dependencies to other processes. Nevertheless, our synthetic manipulation of traces does not depend on the distinction between the two, since we will be adding to the overall delay of an event, i.e., anything other than comp_E.

4.1.4 Feature set and classification. Having regression functions as representations of reference units of execution (phases) allows us to compare new sets of sample data against these functions using goodness-of-fit tests. In our case, coefficient of determination (R^2) and Root-Mean-Square-Deviation (*RMSD*) are taken to quantify and compare the amount of deviation. The comparison considers the collected sample related to a specific input against the reference passport.

The classification algorithms we have are Decision Tree (DT), Random Forest (RF), Gaussian Naïve Bayes (GaussianNB), k-Nearest Neighbours (k-NN), Linear Support Vector Classification (LinearSVC) and Kernel Support Vector Machine (KernelSVM)[1], with the first two having the best results. The feature set involved in the classification and brief descriptions of these features are as follows:

- Operation: Considered phase, i.e., wafer op.
- Metric: Considered metric, i.e., CPU time, cumulative reads (reads count) and cumulative writes (writes count) in our experiments for this use-case
- $D_{\text{pid, phase, event}}(R^2)$: Percentage difference between the R^2 value and the R^2 value of the reference passport per PID and per event, e.g., for a single wafer op. phase, we will have *pid_event* columns such as, *23_compute*, *23_read* and *23_write* for PID 23, corresponding to deviations of computation, write and read behaviour, respectively.
- $D_{\text{pid, phase, event}}(RMSD)$: Percentage difference between the *RMSD* value and the *RMSD* value of the reference passport per PID and per event, e.g., for a single wafer op. phase, we will have *pid_event* columns such as, *23_compute*, *23_read* and *23_write* for PID 23, corresponding to deviations of computation, write and read behaviour, respectively. Note that if a PID is strictly a producer of data, it will not have a read column, or for the case of a strict consumer, a write column will not be present.
- Label: Normal, Anomaly 1, Anomaly 2, etc. (Persistent Benign, Persistent Harmful, Transient Benign and Transient Harmful for this use-case, as will be discussed in Section 5.1)

A total of 66 different processes are involved in this use-case's phase data. We split the data into 70% training and 30% test data. Our data analysis pipeline has been written in Python 3.7 and for our regression and classification needs, we rely on Scikit-learn 0.23.1 machine learning library [36].

4.2 Power passports workflow

Our set-up for a proof-of-concept, specific to the image analysis use-case, alongside the feature engineering and feature extraction pipeline is given in Figure 7. This set-up consists of an ODROID-XU4 computing device, implementing the ARM big.LITTLE computing architecture. We are running a stripped-down Linux distribution and the main running application is a neural network-based image analysis software, detecting if cars are present in images. The platform is capable of receiving images from either a camera, or a storage device and in our case, images are provided via a storage device. Note that for this use-case, the combination of the proof-of-concept set-up and the data processing pipeline has less complexity, mainly due to the absence of a digital twin, i.e., a simulator. Here, there is no need for such a twin, for we are able to include real anomalous conditions, eliminating the requirement for synthesising anomalous traces. There will still be a need for a digital or a physical twin when it comes to evaluating actuation policies though.

[1]The hyperparameters used with Scikit-learn library for each classifier besides the default values are as follows: $criterion = entropy$ for DT; $n\_estimators = 100$ for RF; $n\_jobs = -1$ for k-NN; $max\_iter = 50000$ for LinearSVC; $kernel = poly$ and $degree = 5$ for KernelSVM.

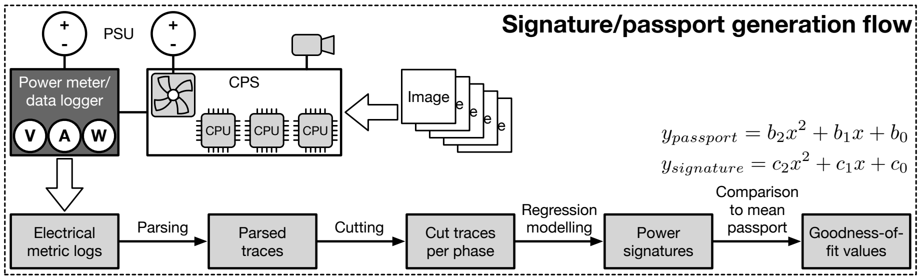


Fig. 7. Our experimental platform, electrical EFB metric collection set-up and the data processing pipeline for power signatures/passports (note the independent power supply to the fan)

For our electrical EFB data collection, we rely on the Otii Arc power data logger unit [37]. The data logger collects electric potential in Volts and electric current in Amps with the sampling rates of 1 kHz and 4 kHz, respectively. Timestamps for each data collection is also recorded alongside these metric values. As shown in Figure 7, data collection is followed by its compartmentalisation (cutting) and application of regression modelling next, resulting in power signatures/passports. The goodness-of-fit values are acquired by comparing a signature to a passport.

Given that we have electrical potential, electrical current and time readings, we consider the three metrics, *current* in Amps, *power* in Watts and *energy* in Milliwatt-hours, for generation of signatures. The image analysis running on the target platform has two main operations, i.e., reading images from a storage and applying a neural network detection algorithm on them. Thus, we have considered the following atomic and combo phases for current, power and energy readings:

- Image op.: An *atomic* phase for the image loading operation,
- Neural op.: An *atomic* phase for the neural network operation,
- Cycle op.: A *combo* phase for a full image cycle, including image loading and neural network operations.

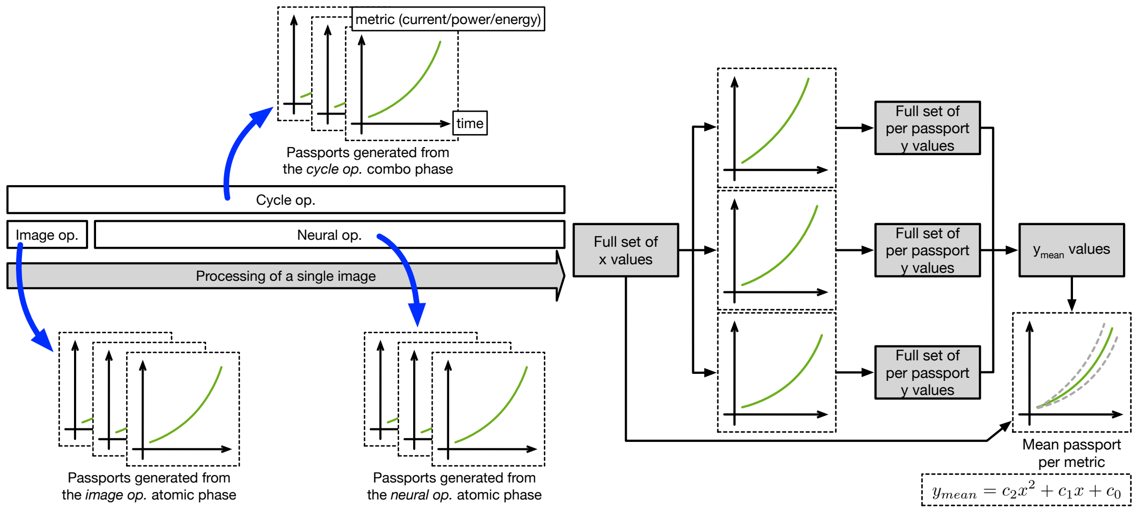
As depicted in Figure 8a, we compartmentalise the processing of an image into execution units, based on the mentioned phases. The parsing is followed by a cutting step, breaking the parsed data per phase.

4.2.1 Mean passports. In addition to individual power passports, we are also generating *mean power passports* as a unified representation for many reference executions with different input data. Mean passports are generated per metric and per phase type. When it comes to mean power passports, their generation is not a straightforward task, as there needs to be matching timestamps. Basically, we are calculating the mean of many regression functions, which can be written as,

$$y_{\text{mean}}(x) = \frac{f(x) + g(x) + h(x) + ...}{n},$$

with x being the independent variable, time, and n the number of functions.

Either we have to do listwise deletion and remove independent variable readings which do not exist in all phases, or we have to perform data imputation. The latter is much more preferable, for there are executions that take slightly longer and we do not wish to disregard valid data collection points residing at the end of longer executions. In principle, this applies to other unmatched points as well, regardless of their location within the execution time frame. We have already generated regression models for data collections as their representations and as such, we can perform regression-based imputation by predicting the dependent variable values (metric) for missing independent variable values. This arrangement is especially convenient, since we already have generated and we use these very same regression models


metric (current/power/energy)
time
Passports generated from the *cycle op.* combo phase
Cycle op.
Image op.
Neural op.
Processing of a single image
Passports generated from the *image op.* atomic phase
Passports generated from the *neural op.* atomic phase
Full set of x values
Full set of per passport y values
Full set of per passport y values
Full set of per passport y values
y_{mean} values
Mean passport per metric
$y_{mean} = c_2x^2 + c_1x + c_0$

(a) Execution phases

(b) Mean passport generation

Fig. 8. (a) Different execution phases for an image processing task, i.e., *atomic image operation*, *atomic neural operation* and *combo cycle operation*, encompassing the first two; (b) Mean passport generation flow using the full set of x values (independent variable) from all available passports for the same metric and generating equal size sets of y values (dependent variable) by means of regression-based imputation

in comparisons and goodness-of-fit tests. There will be no extra bias other than what already exists as part of regression models. Figure 8b depicts this process.

4.2.2 Feature set and classification. Similar to the previous use-case, coefficient of determination (R^2) and Root-Mean-Square-Deviation (*RMSD*) are taken to quantify and compare the amount of deviation. The comparison considers the collected sample related to a specific input against the universal mean passport. Obviously the metric and the phase should be the same. Therefore, there is no need for special considerations on the mean passport leg of the comparison, as there is only one per metric and per phase.

Same classification algorithms as the previous use-case are used, namely, DT, RF, GaussianNB, k-NN, LinearSVC and KernelSVM[2], with the first two having the best results. The feature set involved in the classification and brief descriptions of these features are as follows:

- Operation: Considered phase out of image op., neural op. and cycle op.
- Core type: Utilised CPU core, big or little
- Metric: Considered metric out of current, power and energy consumption
- Execution time: Execution time for the phase (this is the one piece of information, turning the view into grey box, instead of black box, since we have the boundaries of phases in time)
- Coefficient 2: Coefficient for the second degree term, x^2, of the regression function
- Coefficient 1: Coefficient for the first degree term, x, of the regression function
- Intercept: Intercept value of the regression function
- R^2: Goodness-of-fit value for sample points from one image against the reference mean passport

[2]Same hyperparameters as the previous use-case were considered.

- $D_i(R^2)$: Absolute difference between the R^2 value and the R^2 value of the reference mean passport
- $RMSD$: Goodness-of-fit value for sample points from one image against the reference mean passport
- $D_i(RMSD)$: Absolute difference between the $RMSD$ value and the $RMSD$ value of the reference mean passport
- Label: Normal, Anomaly 1, Anomaly 2, etc. (NoFan and UnderVolt for this use-case, as will be discussed in Section 5.2)

Here, we also split the data into 70% training and 30% test data.

5 EXPERIMENTAL SET-UP

Let us go through the actual experiments performed for each use-case. Considering that we are following a data-centric approach, the main goal is to come up with multiple scenarios and relevant data sets to have enough variation for training and testing in our AI-based solution.

5.1 Semiconductor photolithography use-case

In the set of experiments conducted with the ASML semiconductor photolithography machine, we have considered one execution phase corresponding to a repetitive task performed by the machine, wafer processing operation. Photolithography machines are always used for exposing batches of wafers with the same chip design, hence repetitive task. The simulation's output traces corresponding to the baseline execution were used to generate software passports for each process involved in the phase. In this fashion, the simulator, i.e., the digital twin of the system, is being considered as a reference platform.

In order to generate a proper data set containing different types of performance anomalies, we have composed several hiccup injection scenarios. The hiccup injection protocol, resulting in these scenarios is elaborated in the following steps:

(1) From the total number of available processes, 15%, 30% and 45% of them were selected randomly to introduce modifications in their traces.
(2) From the processes selected in the previous step, two main performance anomaly groups, persistent and transient, were created. Software processes in the traces were randomly assigned to each group, while a 10% overlap between the two groups was ensured. For example, if we have the option to inject hiccups in twenty processes and we have to generate two groups with 10% overlap, the resulting groups could look like,

$$\text{Persistent group} = \{0, 1, 2, 5, 7, 8, 10, 11, 12, 13\} \text{ and}$$
$$\text{Transient group} = \{0, 3, 4, 6, 9, 14, 15, 16, 17, 18\}.$$

The overlap percentage is configurable and its purpose is to create a more realistic scenario, where there is no clear separation between the set of processes that could cause persistent anomalies and the ones that could cause transient anomalies.
(3) Computation, read, write, or all types of events are selected from the designated group of processes in the previous step. For each of the event types selected, either 10%, 15%, 30%, or 45% of them are modified, adding some overhead to the event's elapsed time.
(4) The overhead applied to each event is either of 5%, 15%, 30%, or 45% of their total event elapsed time.
(5) Lastly, the above steps were repeated five times to create sufficient data for different categories.

Using the protocol described above, we created 1920 different scenarios, where each scenario contains traces corresponding to the monitored software processes. These modified traces were simulated, using the digital twin environment described in Section 4.1.2, to determine the impact on the phase execution time. Since our potentially anomalous execution scenarios and the relevant data are synthetically generated, we need an indicator to detect if the injected hiccup has turned the execution into an anomalous one. This indicator is the execution time of the phase. While considering the original phase execution time as the baseline, for the 0-2%, 2-4% and above 4% amounts of increase in the phase execution time, we consider the behaviour to be Normal, Benign and Harmful, respectively. Combined with the groups of processes that are used for hiccup injection, we will end up with the categories, *Normal*, *Persistent Benign*, *Persistent Harmful*, *Transient Benign* and *Transient Harmful*.

5.2 Image analysis use-case

The input data for the image analysis application are provided on a storage device. We have considered two different sets of images, first one being proper images with meaningful scenery. This batch includes images with and without a car depicted in them. The second batch includes images that do not depict any particular shape and have purely randomised pixels, introducing variation in the input. In this fashion, we could evaluate if the composition of an image is a factor for our workflow. Each batch is used in two different executions, one involving a single round of image analysis and the second, involving ten rounds of image analysis, meaning the same batch is processed ten times, sequentially. We have chosen the number of rounds arbitrarily and with the aim to have a long enough execution, reflecting the effects of anomalies. Each batch includes 30 images, making the workload for ten rounds of processing as 300 images. We have also executed every combination of conditions twice, by assigning the application to either a big, or a little core on the platform. The list of performed data collections are as follows:

- Case 1: 1 round of execution, regular images, little core (30 total images)
- Case 2: 10 rounds of execution, regular images, little core (300 total images)
- Case 3: 1 round of execution, regular images, big core (30 total images)
- Case 4: 10 rounds of execution, regular images, big core (300 total images)
- Case 5: 1 round of execution, randomised images, little core (30 total images)
- Case 6: 10 rounds of execution, randomised images, little core (300 total images)
- Case 7: 1 round of execution, randomised images, big core (30 total images)
- Case 8: 10 rounds of execution, randomised images, big core (300 total images)

The structuring of our workloads for the aforementioned data collection cases, fits the principle of repetitive task execution for industrial CPS rather well. Keep in mind that although these tasks are repetitive, but the underlying non-determinism is still present, as the behaviour of the system has subtle variations per execution, even with the same exact input. Now that different cases are defined, we have considered two different anomalies, affecting the performance and the reliability of the system.

Malfunction of the cooling system. For this anomaly, henceforth called *NoFan*, we have disabled the cooling fan of the platform's CPU block. Keep in mind that in our set-up, the fan has a separate supply of power to begin with and will not directly affect electrical EFB metric readings of the platform, as depicted in Figure 7.

Unstable power delivery. For this scenario, henceforth called *UnderVolt*, we have reduced the voltage supply to a level below the required amount for the platform, but still keeping the device functional. This was a reduction from 5.0 Volts to 4.7 Volts. We have also made sure that the voltage supply is at a sufficient level and it will not result in glitches during the execution of tasks.

As it was the arrangement for data collection under normal circumstances, reflecting the normal behaviour, we have considered the exact same cases with different batches of images, different numbers of processing rounds and different cores. Accordingly, our experiments resulted in eight cases for each scenario, representing Normal, NoFan and UnderVolt situations. It must be mentioned that having equal number of cases and basically equal number of data fields for all scenarios is advantageous as it will result in a balanced data set for classifiers. Having a balanced data set will eliminate the need for imbalance countering techniques, e.g., undersampling and oversampling.

6 CLASSIFICATION RESULTS

Our results mostly focus on performance anomaly prediction accuracies of various classification algorithms. We provide these separately for our two industrial use-cases. Accurate classification, as the ultimate goal of the workflow from Figure 3, can be achieved with the right choice of metric, execution phase and a suitable classification algorithm.

6.1 Semiconductor photolithography use-case

We have tried Decision Tree (DT), Random Forest (RF), Gaussian Naïve Bayes (GaussianNB), k-Nearest Neighbours (k-NN), Linear Support Vector Classification (LinearSVC) and Kernel Support Vector Machine (KernelSVM) classifiers with the data set representing a single wafer's processing, alongside CPU time, reads count and writes count, as our metrics of choice, as well as the combination of all three. As such, Table 1 presents overall prediction accuracies for different classifiers.

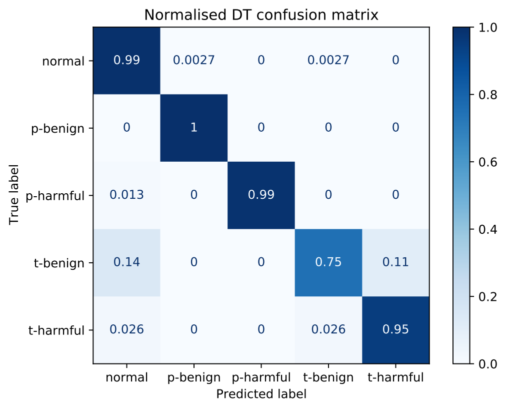
Table 1. Overall prediction accuracy percentage of Decision Tree (DT), Random Forest (RF), Gaussian Naïve Bayes (GaussianNB), k-Nearest Neighbours (k-NN), Linear Support Vector Classification (LinearSVC) and Kernel Support Vector Machine (KernelSVM) classifiers based on the data sets generated from different individual metrics, as well as their combination. The highest accuracies belong to DT and RF, when all three metrics, i.e., CPU time, reads count and writes count, are considered.

Metric	Phase	DT	RF	GaussianNB	k-NN	LinearSVC	KernelSVM
CPU time	wafer op.	96.00	95.65	79.51	92.18	90.45	66.49
reads count	wafer op.	92.01	95.48	32.98	93.57	90.79	75.69
writes count	wafer op.	94.96	95.65	76.90	92.18	86.45	66.49
all three	wafer op.	97.56	95.83	76.38	91.66	90.97	66.49

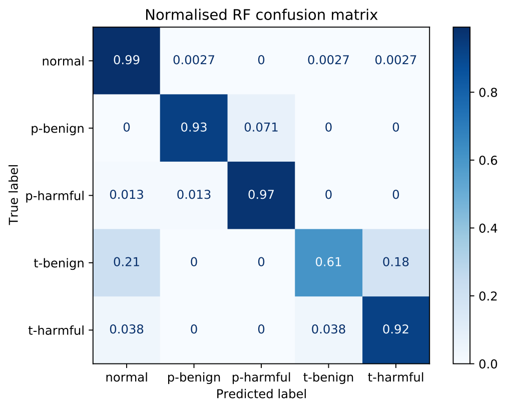
The overall prediction accuracy of DT, RF, GaussianNB, k-NN, LinearSVC and KernelSVM classifiers, while using the data generated from *single wafer processing* phases together with the combination of all three metric, are 97.56%, 95.83%, 76.38%, 91.66%, 90.97% and 66.49%, respectively. The confusion matrices for all classifiers with this combination are depicted in Figure 9, showing the higher prediction performance for DT and RF. As a result of randomness in our synthetic delay injection and scenario generation, the data set has considerable imbalance, leading to different accuracies for different label predictions. With an increased number of scenarios representing less frequent labels, relevant accuracies will improve. This is demonstrated in the image analysis use-case.

6.2 Image analysis use-case

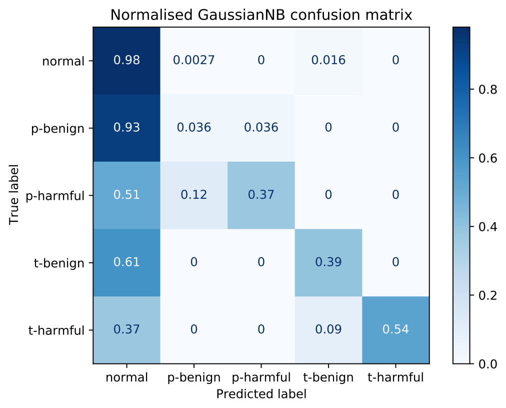
For this use-case, we have also tried DT, RF, GaussianNB, k-NN, LinearSVC and KernelSVM classifiers with different subsets of our data set. We observe that the choice of phase and metric as sources of data has a considerable effect on



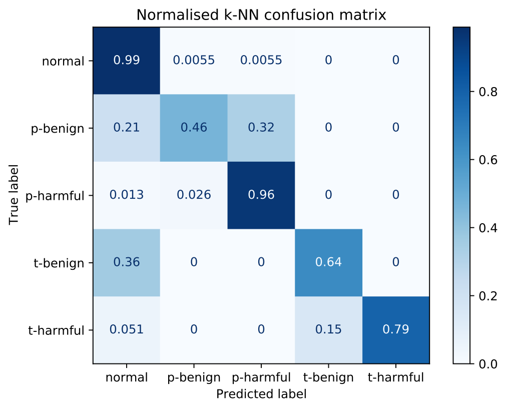
(a) DT classifier



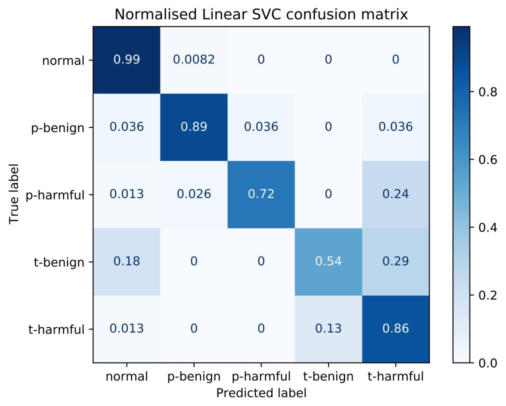
(b) RF classifier



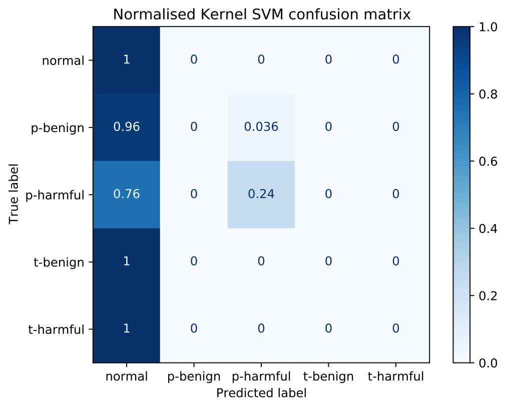
(c) GaussianNB classifier



(d) k-NN classifier



(e) LinearSVC classifier



(f) KernelSVM classifier

Fig. 9. Normalised confusion matrices for (a) DT, (b) RF, (c) GaussianNB, (d) k-NN, (e) LinearSVC and (f) KernelSVM classifiers, considering the anomaly categories, Normal, Persistent Benign (p-benign), Persistent Harmful (p-harmful), Transient Benign (t-benign) and Transient Harmful (t-harmful) as labels, with the data set based on the CPU time metric and the wafer processing phase

the prediction accuracy of the classification. We have tried classifications using data from all three metrics at the same time, as well as every metric individually. For these trials, we have considered either atomic image op., atomic neural op., combo cycle op., or the combination of the data from both atomic image and atomic neural operations. Table 2 presents these choices alongside their resulting overall prediction accuracies.

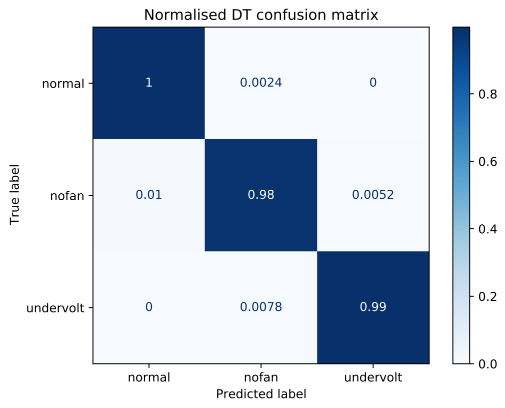
Table 2. Overall prediction accuracy percentage of Decision Tree (DT), Random Forest (RF), Gaussian Naïve Bayes (GaussianNB), k-Nearest Neighbours (k-NN), Linear Support Vector Classification (LinearSVC) and Kernel Support Vector Machine (KernelSVM) classifiers based on data sets generated from different combinations of metrics and phases

Metric	Phase	DT	RF	GaussianNB	k-NN	LinearSVC	KernelSVM
current	image op.	89.83	92.80	67.40	69.34	38.86	39.28
current	neural op.	98.89	98.81	90.09	98.13	91.95	90.51
current	cycle op.	99.15	99.23	90.09	98.22	86.45	89.92
current	image + neural op.	94.23	96.06	57.05	83.94	61.79	38.03
power	image op.	81.79	83.23	60.71	62.06	36.15	37.25
power	neural op.	96.86	96.61	84.25	93.98	70.70	70.78
power	cycle op.	97.03	97.54	87.29	93.22	62.99	70.95
power	image + neural op.	89.70	91.48	60.10	80.26	59.16	36.00
energy	image op.	74.25	77.13	66.04	59.69	69.85	38.01
energy	neural op.	96.69	97.37	82.21	94.83	87.89	89.33
energy	cycle op.	97.29	97.71	83.06	95.25	88.56	89.50
energy	image + neural op.	86.99	88.47	68.91	77.72	78.90	37.06
all three	image op.	83.05	84.89	43.91	62.55	45.41	35.18
all three	neural op.	97.85	98.22	64.92	96.27	61.14	51.36
all three	cycle op.	97.96	98.50	66.05	96.07	68.68	51.42
all three	image + neural op.	89.86	91.52	49.39	79.51	57.56	35.39

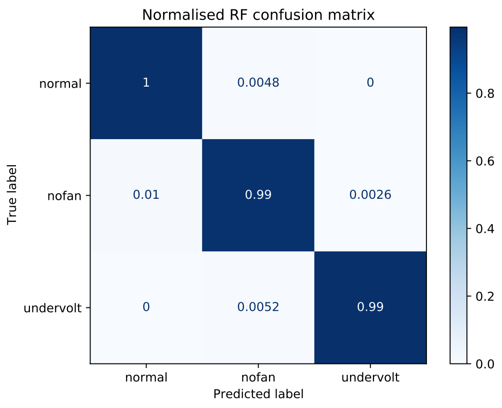
The overall prediction accuracy of DT, RF, GaussianNB, k-NN, LinearSVC and KernelSVM classifiers, while using the data generated from *combo cycle op.* phases together with the electrical current metric, are 99.15%, 99.23%, 90.09%, 98.22%, 86.45% and 89.92%, respectively. Here, DT and RF show the highest accuracies out of all combinations. The confusion matrices for all classifiers with this combination are depicted in Figure 10, again showing the higher prediction performance for DT and RF classifiers. High prediction accuracy is also observable across the board for all labels, as a result of having a balanced data set.

7 ANALYSIS AND DISCUSSION

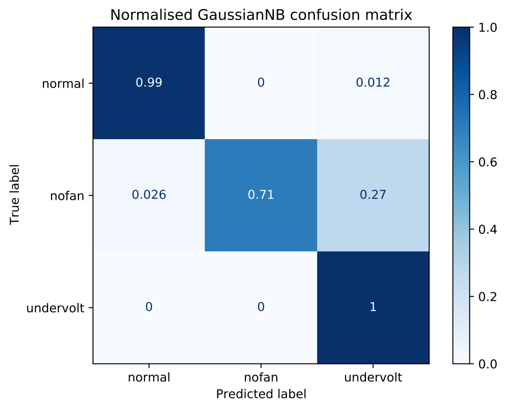
Our methodology's outcome is based on feature engineering and traditional classification algorithms. While the methodology itself is applicable to any industrial CPS, implementations are use-case specific. Accordingly, it is important to fine-tune the workflow based on the platform at hand and its characteristics. Such a fine-tuning is necessary to maximise the classification accuracy and bring out the full potential of the method. Fine-tuning in this context is the process of selecting the best combination of EFB metrics, execution phases and classification algorithms. While EFB metrics and execution phases of choice are highly use-case dependent, we can see that DT and RF classifiers are the best performing ones. In the case of the DT classifier, a simple but powerful analysis is to visualise the DT graph. The graph can guide us to choose the right execution phase, for the more compact and contained the graph, the easier it is for the classifier to distinguish between labels. For our use-cases, we see that the combination of all metrics and the



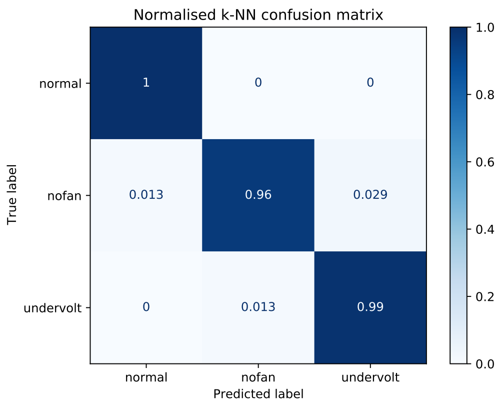
(a) DT classifier



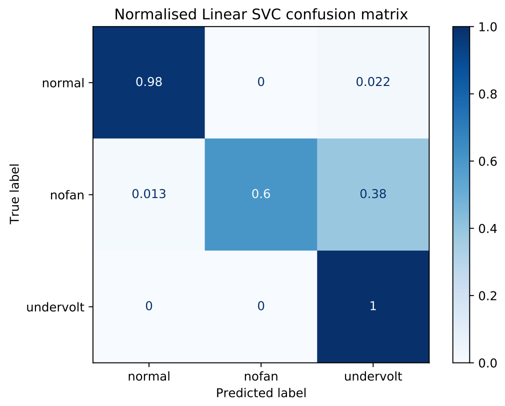
(b) RF classifier



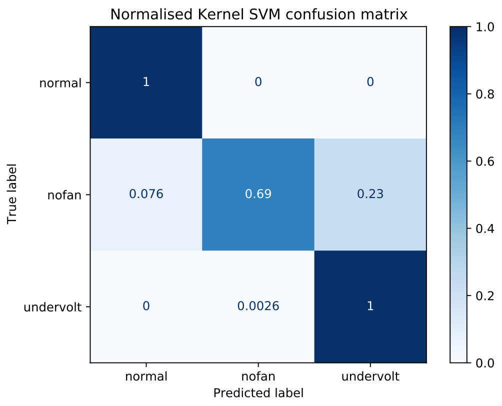
(c) GaussianNB classifier



(d) k-NN classifier



(e) LinearSVC classifier



(f) KernelSVM classifier

Fig. 10. Normalised confusion matrices for (a) DT, (b) RF, (c) GaussianNB, (d) k-NN, (e) LinearSVC and (f) KernelSVM classifiers, considering the anomaly categories, Normal, NoFan and UnderVolt as labels, with the data set put together for the electrical current metric and for the cycle op. phase

wafer processing phase, as well as the choice of electrical current as the metric and the cycle operation phase, lead to rather compact DT graphs, shown in Figures 11 and 12, respectively.

Without diving too deep into the analytical comparison between different classifiers we have experimented with, a few statements can be made. Our motivation behind the use of common traditional ML algorithms is to see which classifiers perform best out-of-the-box, without any hyperparameter tuning effort. As DT and RF are visibly superior for this type of data, we chose to consider these for our complete solution. Out of these two, DT provides certain level of root cause analysis as well, which is another added value, favouring DT. With moderate effort, one can expect to improve the prediction performance of other classifier, but to which extend, it is to be seen. From an analysis point of view however, considering the type of data we deal with and the features we have developed out of it, we can see why some classifiers perform poorly. Let us focus on GaussianNB and KernelSVM classifiers. For instance, the NB classifier is a parametric one. Although our features are generated from samples of metric data and regressors were developed based on these collections, the specifics of these regression functions (coefficients and intercept) and goodness-of-fit measurements are not expected to follow a clear distribution. As such, it is expected that parametric algorithms will not be a suitable choice. In the case of the SVM classifier, SVM is known to be very sensitive towards the choice of kernel and hyperparameter tuning, hence the out-of-the-box disadvantage.

One might argue that anomaly specific monitoring is also capable of detection, especially for metrics external to the system, e.g., voltage supply. While this is a perfectly valid proposition to detect voltage supply instabilities, such detection lacks versatility, as it is single-anomaly specific and assumes prior knowledge of the anomaly type. In comparison, our methodology can detect and differentiate between multiple anomalies. This versatility is a direct result of data-centricity and comparisons based on behavioural signatures. Our methodology also foresees detection of unknown anomalies through described comparison tools, namely, goodness-of-fit values.

7.1 Explainable output

While treating the system under scrutiny as a black box can be advantageous and a requirement in certain use-cases, having a black/grey box solution, i.e., methodology, might not be so favourable. It is important to be able to figure out which changes at every different step of the methodology will lead to a certain result. Since our methodology, as mentioned in Section 2, is based on the combination of extensive feature engineering and traditional classification algorithms, we can reap the benefits of explainable output. More precisely, we can backtrack our choices at different data manipulating steps and highlight the accuracy-increasing ones. While such a capability can help with optimising the workflow itself, e.g., by choosing better metrics and phases as the sources of data, it could also help in narrowing down the root causes of anomalies. For instance, the specific PIDs responsible for deviations in a phase with multiple active PIDs can be detected, as signatures and passports are generated per PID. As an example, if we look at the graph from Figure 11 and traverse the tree from its root as,

(1) True, False, False, and
(2) False, True, False,

both paths lead to *transient harmful* anomaly classification. For the enumeration 1, we see that software signatures from processes 44, 34 and 41 in combination with metrics writes count, CPU time and reads count, respectively, are definitive for the classifier. This knowledge provides a considerable reduction for the scope of desired analyses. Now we know which signatures and in turn, which parts of the raw traces for a phase have resulted in a *transient harmful* anomaly. For the enumeration 2, we observe that the decision is based on the software signatures from processes 41

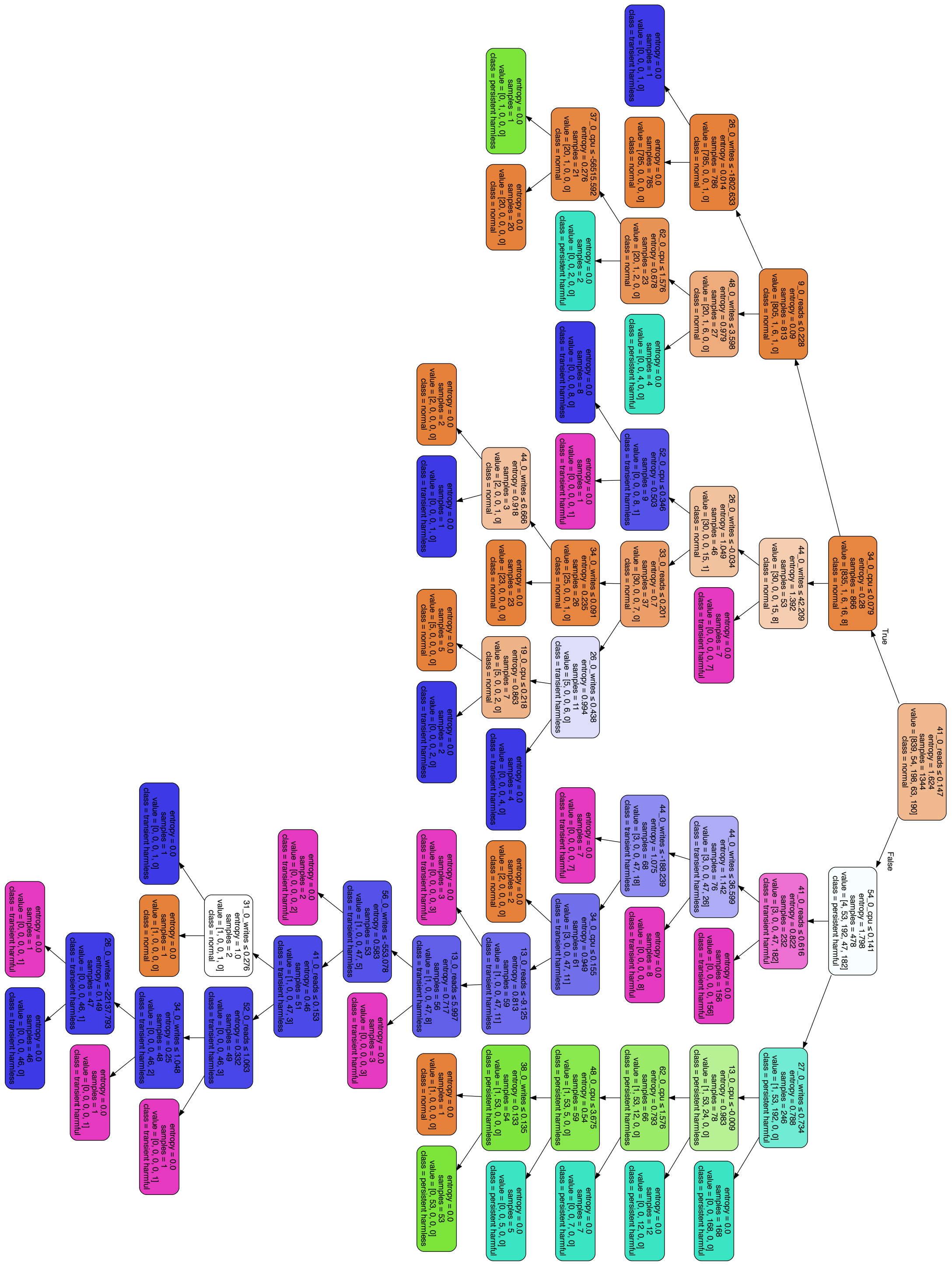


Fig. 11. Showcasing the compactness of Decision Tree (DT) graph for the semiconductor photolithography machine use-case, considering all three metrics, during the wafer op. phase.

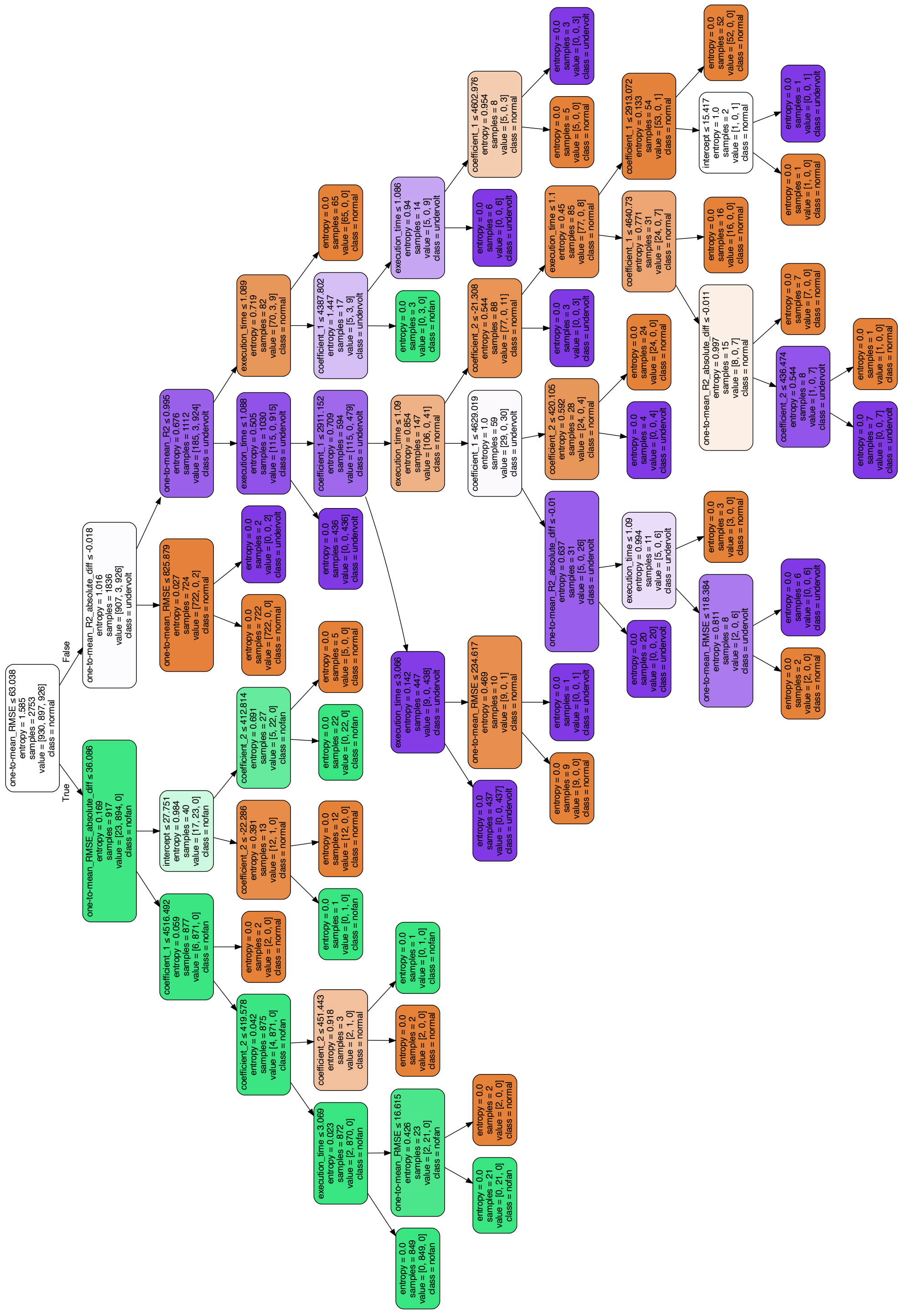


Fig. 12. Showcasing the compactness of Decision Tree (DT) graph for the image analysis platform use-case, considering current as the metric, during the cycle op. phase.

and 54, in combination with the metric reads count for process 41, twice, and the metric CPU time for process 54. Note that there are more leaves with this label present.

The same argument holds for metrics as well, i.e., upon presence of multiple metrics in the training data set and the inference data, we can narrow down the impact of anomaly at hand to a specific metric or a subset of considered metrics. Such knowledge is rather informative for any expert, looking for the actual root cause and it is a strong motivation for having passports and signatures per PID, per phase and per metric. Although the ideal root cause analysis workflow is a fully automatic one, the discussed examples demonstrate the usefulness of semi-automatic or methodically gathered inherent insights, which is a form of Decision Support System (DSS). The use of DSS is prevalent in a wide range of contexts nowadays [1, 12, 42, 41].

In our context and considering the use of our data processing pipeline in combination with a DT classifier, explainability is inherent to the solution. A solution such as ours can be considered as a DSS for experts analysing the system behaviour for root cause analysis. Though the identified anomaly can be backtracked, revealing the subset of PIDs, metrics and phases relevant to the label, the expert still has to utilise system knowledge to figure out the actual fault, e.g., lack of a certain resource resulting in a delay, or an incorrect input resulting in a spike in CPU time measurement. The reach of our solution's visibility of the system ends at the boundary of data collection, generating the raw sensory input to our data processing pipelines.

Besides root cause detection, other design-improving intelligence is available too. For instance, the methodology can show if a specific batch of inputs to the system results in unexpected anomalous behaviour, revealing design deficiencies in handling specific categories of inputs.

7.2 Limitations of the study

We have had a few limitations when dealing with our use-cases. Some of these were imposed by the industrial platform under scrutiny, such as the absence of real anomalous scenarios in the case of the semiconductor photolithography machine. Others were self-imposed with the goal of having better control over experimentation and development of the methodology. For instance, in the case of the image analysis platform, we have opted for a single-threaded and single process application. The choice is still in line with real-world conditions as such set-ups are the preferred choice in low-power use-cases.

8 RELATED WORK

Anomaly detection and identification have been on the agenda of the research community for a long period. The relevant body of knowledge is fairly large, but is also covered rather well in two extensive survey papers. Both Chandola et al. [6] and Ibidunmoye et al. [20] give collections of papers, diving into different aspects of anomaly detection. The challenging nature of decision support, leading to actuations and better designs for industrial systems, is also attested by the grand challenges presented by Fowler [14]. Most of the body of work given in [20] focus on performance anomalies in distributed systems [19], cloud environments [16] and web applications [8], whereas our methodology is specifically tailored towards industrial CPS. We are of the opinion that our approach helps in simplifying the task for repetitive systems by considering execution phases, regression-based representations and more importantly, white box or black/grey box views of industrial CPS, where it fits. We have also made use of a digital twin for our first use-case, which is in line with the ever-growing importance of such virtual representations when it comes to complex CPS [38].

The very recent publication by Luo et al. [28] provides a handy listing of surveys on anomaly detection. Out of the given list, surveys by Giraldo et al.[17], Mitchell et al. [31] and Lun et al. [46], specifically focus on anomaly detection for CPS. Interestingly enough, these surveys all focus on anomalies to address security and attack detection. This has been a common trend, to see anomalies as consequences of security incidents, which is a valid motivation. Industrial CPS such as photolithography machines on the other hand, are totally isolated and there is more added value in tackling performance anomalies to improve the machine yield, which our work tries to address.

Electrical metrics, especially electrical power analysis, have a profound role in cybersecurity research. The famous and now classic paper by Kocher et al. [24] is a great example. What such publications have in common is their view towards electrical power in particular and electrical metrics in general, which is seen as a source of side-channel information. In principle, our view is rather similar and we see such metrics, external to the system, as reflections of its functional behaviour, which is the definition of EFB. Further uses of power signals for anomaly detection in the scope of cybersecurity are given in [4, 23, 27, 45].

Considering our behavioural signature/passport generation technique, the common use of regression models as a statistical tool for data estimation and inference is well argued in literature [21, 7]. This is especially true for different performance parameters of application processes as Lee and Brooks suggest [25]. Lee et al. also employ the notion of piecewise polynomial regression [26]. We have conducted a comparable strategy by dividing a timeline into meaningful phases, based on drastic changes in the values of CPU utilisation. Though, our division criteria aims at having meaningful and repeatable phases. We are not using all points from an execution, i.e., certain unsuitable parts are being omitted.

The survey by Chandola et al. [6] includes classification algorithms from other works. These algorithms are applicable to repetitive data such as ours. We have specifically focused on DT [40], RF [3], GaussianNB [15], k-NN [10], LinearSVC [2] and KernelSVM [9] as traditional classifiers. On the other hand, the complexity of modern CPS has driven researchers towards Deep Learning (DL) techniques more than ever. In particular, more recent publications [18, 28, 43, 39] demonstrate this tendency. The recent survey by Chalapathy and Chawla [5], as well as the paper by Ratasich et al. [39], do point out the differences between traditional Machine Learning (ML) and sophisticated DL techniques by mentioning the black box nature of DL models. One advantage of our methodology, which is based on traditional ML, is the ability to traverse the resulting models and backtrack the data pipeline, as mentioned in Section 7.1. When dealing with anomalies, root cause analysis is arguably an integral requirement. Traditional ML in general and our method in particular is rather capable in this aspect.

9 CONCLUSION AND FUTURE WORK

We have shown that a well-devised data-centric solution, as we have presented, is capable of anomaly detection and classification for industrial CPS, with high accuracy. Such a data-centric solution is much more flexible than self-adaptivity based on traditional control mechanisms, as it is capable of consuming monitoring data from a multitude of sources. More precisely, we have shown a behaviour classification methodology composed of Extra-Functional Behaviour (EFB) monitoring at runtime, compartmentalisation of execution timeline into repetitive execution phases, generation of representative behavioural signatures and passports, deviation quantification based on goodness-of-fit tests, and traditional classification algorithms.

Our behavioural signature construct is an especially convenient tool, accommodating a variety of metrics as input and representing arbitrary lengths of execution timeline of a system in a compact fashion. This compactness results in easy to store representations of reference behaviour, i.e., passport, as well as efficient to compare representations of ongoing behaviour, i.e., signatures. Using signatures, behaviour can be represented per execution phase, per metric and per process, if the application consists of multiple processes. We have also shown that the resulting comparison data as a feature set, is most suitable for *decision tree* and *random forest* classifiers, achieving accuracies as high as 99% in certain set-ups. The feature set is complete enough to facilitate classification of anomalies early on. In most anomalous cases it will take a while before visible deviations are present, e.g., under NoFan conditions.

In view of our proof-of-concept use-cases, we have demonstrated that our methodology is valid throughout the industrial CPS complexity spectrum. We have implemented our methodology for a semiconductor photolithography machine as a large and complex industrial CPS, as well as an image analysis platform as a low-power and less complex industrial CPS. We have also described different monitoring techniques. Techniques using system-level metrics in a white box, invasive and communication-centric approach, and techniques using external metrics in a grey box, passive and side-channel-oriented approach. We have presented the role of a digital twin for industrial CPS and elaborated its creation using high-level modelling and event-based simulation of traces captured with communication-centric monitoring.

Future work. There are a few leads in our study that can be followed as part of a future work. The application of more advanced AI algorithms, i.e., neural networks and the implications of the deployment is an interesting aspect. We would like to see how they compare to our current method, as they will drastically reduce the explainable output characteristic. Neural networks, e.g., Convolutional Neural Networks (CNN), are capable of extracting features on their own. However, these features are not the same as what we are generating within our pipelines. The input data to a neural network still has to be prepared and formatted, which may require moderate computational effort. One of the important requirements for us is the knowledge of execution phase start and end times. A further step in the automation of our workflow can be achieved by using Change Point Detection (CPD) for this goal. Development of multidimensional behavioural signatures and passports to represent the behaviour even more uniquely is also on our agenda. In this fashion, multiple metrics are to be used in construction of our regression-based representations. We also would like to experiment with hardware performance counters as EFB metrics in our methodology.

Last but not least, complementing our workflow with the ability to counter the effects of anomalies in industrial CPS will elevate our methodology to an EFB managing solution. This goal is our ongoing research. We hope that our contributions would be a step towards the integration of EFB managing data nodes as common building blocks in industrial CPS, for we believe such data-centric nodes provide multifaceted benefits. These benefits span from anomaly detection and prediction to base design improving intelligence, as discussed throughout this paper.

ACKNOWLEDGMENTS

This paper is composed as part of the research project 14208, titled *"Interactive DSL for Composable EFB Adaptation using Bi-simulation and Extrinsic Coordination (iDAPT)"*, funded by The Netherlands Organisation for Scientific Research (NWO).

REFERENCES

[1] Anna Markella Antoniadi, Yuhan Du, Yasmine Guendouz, Lan Wei, Claudia Mazo, Brett A. Becker, and Catherine Mooney. 2021. Current challenges and future opportunities for xai in machine learning-based clinical decision support systems: a systematic review. *Applied Sciences*, 11, 11. DOI: 10.3390/app11115088.

[2] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. 2002. Support vector clustering. *Journal of Machine Learning Research*, 2, (Mar. 2002), 125–137.

[3] L. Breiman. 2001. Random forests. *Machine Learning*, 45, 1, 5–32. DOI: 10.1023/A:1010933404324.

[4] L. Caviglione, M. Gaggero, J. Lalande, W. Mazurczyk, and M. Urbański. 2016. Seeing the unseen: revealing mobile malware hidden communications via energy consumption and artificial intelligence. *IEEE Transactions on Information Forensics and Security*, 11, 4, 799–810. DOI: 10.1109/TIFS.2015.2510825.

[5] R. Chalapathy and S. Chawla. 2019. Deep learning for anomaly detection: a survey. (2019). arXiv: 1901.03407 [cs.LG].

[6] V. Chandola, A. Banerjee, and V. Kumar. 2009. Anomaly detection: a survey. *ACM Comput. Surv.*, 41, 3, Article 15, (July 2009), 58 pages. DOI: 10.1145/1541880.1541882.

[7] C. Chatfield. 1995. Model uncertainty, data mining and statistical inference. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 158, 3, 419–466. DOI: 10.2307/2983440.

[8] L. Cherkasova, K. Ozonat, M. Ningfang, J. Symons, and E. Smirni. 2008. Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, 452–461. DOI: 10.1109/DSN.2008.4630116.

[9] C. Cortes and V. Vapnik. 1995. Support-vector networks. *Machine Learning*, 20, 3, 273–297. DOI: 10.1007/BF00994018.

[10] T. Cover and P. Hart. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13, 1, 21–27. DOI: 10.1109/TIT.1967.1053964.

[11] P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli. 2012. Modeling cyber-physical systems. *Proceedings of the IEEE*, 100, 1, 13–28. DOI: 10.1109/JPROC.2011.2160929.

[12] Yuhan Du, Anthony R Rafferty, Fionnuala M McAuliffe, Lan Wei, and Catherine Mooney. 2022. An explainable machine learning-based clinical decision support system for prediction of gestational diabetes mellitus. *Scientific Reports*, 12, 1, 1–14. DOI: 10.1038/s41598-022-05112-2.

[13] 2009. *Nasa study on flight software complexity. AIAA Infotech@Aerospace Conference.* DOI: 10.2514/6.2009-1882.

[14] J. W. Fowler and O. Rose. 2004. Grand challenges in modeling and simulation of complex manufacturing systems. *SIMULATION*, 80, 9, 469–476. DOI: 10.1177/0037549704044324.

[15] N. Friedman, D. Geiger, and M. Goldszmidt. 1997. Bayesian network classifiers. *Machine Learning*, 29, 2, 131–163. DOI: 10.1023/A:1007465528199.

[16] S. Fu. 2011. Performance metric selection for autonomic anomaly detection on cloud computing systems. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, 1–5. DOI: 10.1109/GLOCOM.2011.6134532.

[17] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell. 2018. A survey of physics-based attack detection in cyber-physical systems. *ACM Comput. Surv.*, 51, 4, Article 76, (July 2018), 36 pages. DOI: 10.1145/3203245.

[18] J. Goh, S. Adepu, M. Tan, and Z. S. Lee. 2017. Anomaly detection in cyber physical systems using recurrent neural networks. In *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, 140–145. DOI: 10.1109/HASE.2017.36.

[19] D. Gunter, B. L. Tierney, A. Brown, M. Swany, J. Bresnahan, and J. M. Schopf. 2007. Log summarization and anomaly detection for troubleshooting distributed systems. In *2007 8th IEEE/ACM International Conference on Grid Computing*, 226–234. DOI: 10.1109/GRID.2007.4354137.

[20] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth. 2015. Performance anomaly detection and bottleneck identification. *ACM Comput. Surv.*, 48, 1, Article 4, (July 2015), 35 pages. DOI: 10.1145/2791120.

[21] R. Jain. 1990. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley & Sons. ISBN: 978-0-471-50336-1.

[22] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. 2003. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91, 1, 145–164. DOI: 10.1109/JPROC.2002.805824.

[23] H. Kim, J. Smith, and K. G. Shin. 2008. Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services* (MobiSys '08). Association for Computing Machinery, Breckenridge, CO, USA, 239–252. ISBN: 9781605581392. DOI: 10.1145/1378600.1378627.

[24] P. Kocher, J. Jaffe, and B. Jun. 1999. Differential power analysis. In *Advances in Cryptology — CRYPTO' 99.* M. Wiener, (Ed.) Springer Berlin Heidelberg, Berlin, Heidelberg, 388–397. ISBN: 978-3-540-48405-9. DOI: 10.1007/3-540-48405-1_25.

[25] B. C. Lee and D. M. Brooks. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *SIGOPS Oper. Syst. Rev.*, 40, 5, (Oct. 2006), 185–194. DOI: 10.1145/1168917.1168881.

[26] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee. 2007. Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (PPoPP '07). Association for Computing Machinery, San Jose, California, USA, 249–258. ISBN: 978-1-59593-602-8. DOI: 10.1145/1229428.1229479.

[27] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu. 2016. On code execution tracking via power side-channel. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (CCS '16). Association for Computing Machinery, Vienna, Austria, 1019–1031. ISBN: 9781450341394. DOI: 10.1145/2976749.2978299.

[28] Y. Luo, Y. Xiao, L. Cheng, G. Peng, and D. Yao. 2021. Deep learning-based anomaly detection in cyber-physical systems: progress and opportunities. *ACM Comput. Surv.*, 54, 5, Article 106, (May 2021), 36 pages. DOI: 10.1145/3453155.

[29] H. Meyer, U. Odyurt, A. D. Pimentel, E. Paradas, and I. Gonzalez Alonso. 2020. An analytics-based method for performance anomaly classification in cyber-physical systems. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (SAC '20). Association for Computing Machinery, Brno, Czech Republic, 210–217. ISBN: 9781450368667. DOI: 10.1145/3341105.3373851.

[30] H. Meyer, U. Odyurt, S. Polstra, E. Paradas, I. Gonzalez Alonso, and A. D. Pimentel. 2018. On the effectiveness of communication-centric modelling of complex embedded systems. In *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, 979–986. ISBN: 978-1-7281-1141-4. DOI: 10.1109/BDCloud.2018.00143.

[31] R. Mitchell and I. R. Chen. 2014. A survey of intrusion detection techniques for cyber-physical systems. *ACM Comput. Surv.*, 46, 4, Article 55, (Mar. 2014), 29 pages. DOI: 10.1145/2542049.

[32] L. Monostori et al. 2016. Cyber-physical systems in manufacturing. *CIRP Annals*, 65, 2, 621–641. DOI: 10.1016/j.cirp.2016.06.005.

[33] U. Odyurt, H. Meyer, A. D. Pimentel, E. Paradas, and I. Gonzalez Alonso. 2019. Software passports for automated performance anomaly detection of cyber-physical systems. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*. D. N. Pnevmatikatos, M. Pelcat, and M. Jung, (Eds.) Springer International Publishing, Cham, 255–268. ISBN: 978-3-030-27562-4. DOI: 10.1007/978-3-030-27562-4_18.

[34] U. Odyurt, H. Meyer, S. Polstra, E. Paradas, I. Gonzalez Alonso, and A. D. Pimentel. 2018. Work-in-progress: communication-centric analysis of complex embedded computing systems. In *2018 International Conference on Embedded Software (EMSOFT)*, 1–3. ISBN: 978-1-5386-5560-3. DOI: 10.1109/EMSOFT.2018.8537189.

[35] U. Odyurt, J. Roeder, A. D. Pimentel, I. Gonzalez Alonso, and C. de Laat. 2021. Power passports for fault tolerance: anomaly detection in industrial cps using electrical efb. In *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, 152–157. ISBN: 978-1-7281-6207-2. DOI: 10.1109/ICPS49255.2021.9468262.

[36] F. Pedregosa et al. 2011. Scikit-learn: machine learning in python. *Journal of Machine Learning Research*, 12, (Nov. 2011), 2825–2830.

[37] Qoitech AB. 2020. Otii arc - otii by qoitech. (2020). Retrieved June 29, 2021 from www.qoitech.com/otii/.

[38] A. Rasheed, O. San, and T. Kvamsdal. 2020. Digital twin: values, challenges and enablers from a modeling perspective. *IEEE Access*, 8, 21980–22012. DOI: 10.1109/ACCESS.2020.2970143.

[39] D. Ratasich, F. Khalid, F. Geissler, R. Grosu, M. Shafique, and E. Bartocci. 2019. A roadmap toward the resilient internet of things for cyber-physical systems. *IEEE Access*, 7, 13260–13283. DOI: 10.1109/ACCESS.2019.2891969.

[40] O. Maimon and L. Rokach, (Eds.) 2005. *Decision trees. Data Mining and Knowledge Discovery Handbook*. Springer US, Boston, MA, 165–192. ISBN: 978-0-387-25465-4. DOI: 10.1007/0-387-25465-X_9.

[41] Riccardo Rosati, Luca Romeo, Gianalberto Cecchini, Flavio Tonetto, Paolo Viti, Adriano Mancini, and Emanuele Frontoni. 2022. From knowledge-based to big data analytic model: a novel iot and machine learning based decision support system for predictive maintenance in industry 4.0. *Journal of Intelligent Manufacturing*, 1–15. DOI: 10.1007/s10845-022-01960-x.

[42] Riccardo Rosati, Luca Romeo, Carlos Alfaro Goday, Tullio Menga, and Emanuele Frontoni. 2020. Machine learning in capital markets: decision support system for outcome analysis. *IEEE Access*, 8, 109080–109091. DOI: 10.1109/ACCESS.2020.3001455.

[43] P. Schneider and K. Böttinger. 2018. High-performance unsupervised anomaly detection for cyber-physical system networks. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy* (CPS-SPC '18). Association for Computing Machinery, Toronto, Canada, 1–12. ISBN: 9781450359924. DOI: 10.1145/3264888.3264890.

[44] A. Varga and R. Hornig. 2008. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops* (Simutools '08) Article 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Marseille, France, 10 pages. ISBN: 9789639799202.

[45] A. Xu, Y. Jiang, Y. Cao, G. Zhang, X. Ji, and W. Xu. 2019. Addp: anomaly detection for dtu based on power consumption side-channel. In *2019 IEEE 3rd Conference on Energy Internet and Energy System Integration (EI2)*, 2659–2663. DOI: 10.1109/EI247390.2019.9062014.

[46] Y. Zacchia Lun, A. D'Innocenzo, F. Smarra, I. Malavolta, and M. D. Di Benedetto. 2019. State of the art of cyber-physical systems security: an automatic control perspective. *Journal of Systems and Software*, 149, 174–216. DOI: 10.1016/j.jss.2018.12.006.