Highlights

LTL-Based Runtime Verification Framework For Cyber-Attack Anomaly Prediction In Cyber-Physical Systems.

Ayodeji James Akande, Zhe Hou, Ernest Foo, Qinyi Li

- We introduce a cybersecurity framework that leverages runtime verification, integrating data analytics and Linear Temporal Logic (LTL) formula acquisition, to predict anomalies effectively.
- Our approach entails processing historical data, performing clustering or classification to extract anomaly-indicative patterns, and forming data sequences that represent the trend of these patterns.
- A key feature of the framework is its ability to predict anomalies based on temporary patterns observed in data, whether originating from faults or malicious attacks, before their actual manifestation.
- The framework's approach involves extracting LTL formulas from historical data through clustering to identify anomaly-indicative patterns, forming data sequences to represent these patterns, and using an LTL learning algorithm to generate an LTL formula for the detected pattern.

LTL-Based Runtime Verification Framework For Cyber-Attack Anomaly Prediction In Cyber-Physical Systems.

Ayodeji James Akande, Zhe Hou, Ernest Foo, Qinyi Li

[a]*School of Information and Communication Technology, Griffith University, Australia,*

Abstract

An anomaly is any unexpected or abnormal behaviour, event, or data pattern within a network of physical and computational components caused by data errors, cyber-attacks, hardware failures, or other unforeseen events. Anomaly detection analyses events after they occur, while anomaly prediction forecasts them before they manifest. The increasing complexity of Cyber-Physical Systems (CPS) presents challenges in fault management and vulnerability to advanced attacks, highlighting the need for early intervention through anomaly prediction. Existing anomaly prediction methods often fail due to a lack of formal guarantees required for safety-critical applications. In this paper, we introduce our anomaly prediction framework which merges the advantages of data analytics and the derivation of Linear Temporal Logic (LTL) formulas. LTL-based runtime monitoring and checking is a well-established technique efficient for tackling challenges in real-time and promptly. The framework processes historical data, clusters them to extract predictive patterns, and forms data sequences that represent these trends. These sequences are fed into an LTL learning algorithm to produce a formula that represents the pattern. This formula functions as a security property programmed into a runtime checker to verify system correctness and predict the possibility of anomalies. We evaluated our framework using three datasets collected from a cyber-physical system testbed and the experimental findings demonstrate a minimum accuracy of 90% in predicting anomalies.

Keywords: Runtime verification, Linear temporal logic, Cyber-physical systems, Cybersecurity, Critical Infrastructure.

1. Introduction

In modern systems, critical infrastructure is managed digitally by utilising cyber-physical systems (CPS), the metaverse, satellite communication systems, and the Internet of Things (IoT), all of which hold substantial importance in industrial operations and human existence. Although these technologies have significantly improved the management of critical infrastructure, the incidence of cyber-attacks has been steadily increasing and may continue to increase [5, 32, 33]. This trend can be attributed to the growing adoption of digital technologies. Incidents targeting critical infrastructure, encompassing domains such as industrial automation, smart grids, smart cities, autonomous vehicles, and agricultural precision, can potentially produce devastating outcomes for individuals and assets. This paper focuses on Cyber-physical systems (CPS). CPS is a generation of systems that combine computational and physical capabilities, enabling interaction with humans through various new modalities [6]. The physical and software components are closely connected, capable of operating on different spatial and temporal scales, demonstrating diverse behavioural modes, and interacting with each other in context-dependent ways [23].

Shown in Figure 1 is a typical structure of a CPS. In cyber-physical systems (CPS), engineering data (numerical data) typically originate from sensors, actuators, and various devices embedded in physical components. These sensors collect real-world information, and the data is then processed and utilized in the digital realm to enable control and monitoring of the physical system.

1.1. Motivation

Anomalies in CPS can manifest in various ways, such as increased variability and outliers, often indicating deviations from normal operational behaviour. If sensor or actuator data are not reliable, it can cause a system failure. For example, cyber attacks on smart power include GPS spoofing, load redistribution, automatic generation control (AGC) manipulation, line-outage masking, Stuxnet-like attacks targeting sensor measurements, denial of service (DoS), and Byzantine attacks, while attacks targeting actuator commands include Aurora and pricing attacks [47]. Data manipulation is one of the common cyber-attacks that involves altering or modifying engineering data to achieve a specific goal, which can range from extracting valuable information to cause disruption or deception, changing data values,

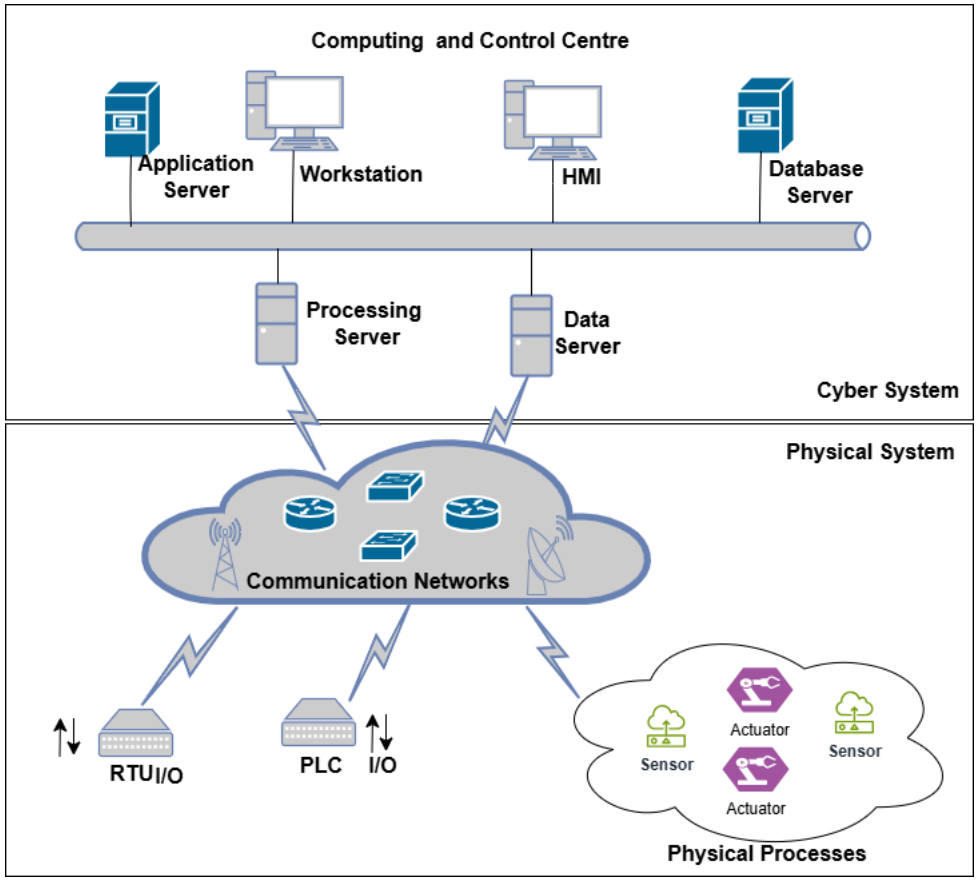


Figure 1: Typical example of Cyber-Physical System

modifying data structures, or tampering with data at various stages of its lifecycle thereby deviating from the expected normal values. This can be termed a data anomaly, also known as an outlier [1]. Bashendy et al. [8] stated that "CPS are typically mission-critical systems, they are intolerant to errors or delays, which can cause catastrophic human, economic, and financial loss if no proper detection and response mitigation mechanisms are applied". Safeguarding cyber-physical systems has become critical, and this necessitates that anomalies be predicted in real-time to maintain minimal impact on its operation.

Yin et al. [46] described anomaly monitoring in time series as a common example of data stream applications and can be categorized into two main types: detecting anomalies in current or historical data and predict-

ing potential future anomalies. The paper further explained that detecting only current or past anomalies comes with practical limitations, which can be overcome by incorporating anomaly prediction. Anomaly prediction is a process of identifying and anticipating unusual or unexpected patterns in data that deviate significantly from typical behaviour. Unlike anomaly detection, which identifies unusual patterns *after* they occur, anomaly prediction seeks to detect anomalies *before* they happen. According to Yin et al. [46], anomaly detection through forecasting is considered a method to predict an unusual point or single instance within a dataset that stands out due to its attributes. Predicting anomalies on time before their impact prevents potential accidents and economic losses [12], and can protect the CPS from costly downtime and disruptions [25].

Despite extensive research in Cyber-Physical Systems (CPS), real-time anomaly prediction for cybersecurity has received less attention due to various challenges. "It is widely held that debugging CPS is challenging; many strongly held beliefs exist regarding how CPS is currently debugged and tested and the suitability of various techniques" [48]. A major challenge in CPS is the limited physical access for personnel to monitor critical assets, which weakens cybersecurity, especially in cyberspace. To proffer solutions to these challenges, prediction based on data analytics can be introduced. Furthermore, in safety-critical applications like aerospace, avionics, and railways, the use of formal methods is crucial for both requirements specification and design validation [13]. In this paper, we introduce a framework founded on runtime verification, which amalgamates the advantages of data analytics and the acquisition of Linear Temporal Logic (LTL) formulas, a formal method for specification and validation. We propose a runtime verification framework based on LTL that predicts anomalies with minimal computational overhead and incorporates a special alert state for rapid mitigation. Data anomalies in time series refer to values that deviate from the anticipated values at specific time points [35]. Our focus is to collate system historical data to analyse and identify predictive patterns leading to an anomaly.

1.2. Literature gap

Machine learning algorithms like Isolation Forest, Decision Trees, Recurrent Neural Networks (RNN), and Long Short-Term Memory (LSTM) are commonly employed for detecting anomalies in time series data. However, they encounter challenges when directly applied to real-time anomaly prediction. Models using RNN and LSTM often encounter challenges, such as high

memory requirements for training [37], deep learning models often require significant computational resources which can lead to high costs when they are used for real-time predictions [11]. Most importantly, machine learning algorithms can classify data to predict anomalies but cannot generate alerts or provide timely responses to anomalies in real-time. To address this challenge, studies [25, 27, 40, 46] have proposed anomaly prediction approaches. However, these approaches are limited by the absence of formal guarantees needed for safety-critical applications. We propose a framework that utilises system-specific derived LTL to predict the anomalies. By using LTL, we designed a framework that is explainable, as LTL specifications are human-readable, deterministic, and effectively capture temporal relationships, making it a preferred choice over existing frameworks. Our framework seamlessly integrates data analytics with the learning of Linear Temporal Logic (LTL) formulas. With runtime verification, the framework checks the system behaviour in real-time to verify that it conforms to defined LTL properties. It helps predict and respond to violations, enhancing the reliability and safety of CPS by continuously checking the desired system properties during operation. Although our framework combines Linear Temporal Logic (LTL) with machine learning, it overcomes the issues associated with using machine learning by deploying the algorithm offline to learn the LTL formula, rather than implementing it in real-time as existing frameworks do.

1.3. Our Framework

Yin et al. [46] explained that time series-based anomaly monitoring can be divided into detecting current or historical anomalies and predicting future ones. While detecting current or past anomalies has its limitations, predicting anomalies in advance can help address the challenges. Predicting anomalous behaviour in industrial assets based on sensor reading represents a key focus in modern business practice and prediction techniques are used to estimate the likelihood that future data will be abnormal [25]. While various studies have explored anomaly prediction methods using machine learning, we go beyond merely applying these methods. Our anomaly prediction framework adopts linear temporal logic as the predictor. This approach not only predicts anomalies but also includes a built-in alert function that aids in anomaly mitigation. Leveraging historical data, the process involves dataset pre-processing and partitioning into training and testing sets. Using the framework, temporal patterns for anomaly prediction are learned from

the training dataset, and the effectiveness of the prediction is then evaluated using the testing dataset.

The framework targets time series data which is composed of individual data points. A key aspect of the framework's process is identifying different patterns within the time series, such as normal, predictive, and anomalous patterns. Normal patterns represent the expected behaviour of a dataset over time. The pattern serves as the crucial baseline data that predictive models rely on to generate accurate forecasts. Any shifts in normal behaviour necessitate updates to the predictive patterns. Data points that fall within the expected range are labelled as "normal data points." Predictive patterns are identified trends or signals in the data that suggest future behaviour or outcomes. These are those sequences of data that could lead to anomalies and are labelled as the "precursor data points". Predictive patterns use normal patterns to anticipate and identify deviations, thereby detecting anomalies. In the real world, there may be patterns that are close to the normal data distribution but are small in size, which can lead to anomalous patterns. We describe these patterns as predictive, as they contribute to deviations from the normal pattern. Anomaly patterns signify unusual behaviour that deviates significantly from normal patterns. These patterns can indicate potential issues, faults, or unexpected events. Anomalies are characterized by values that fall outside the expected range established by normal patterns, often appearing as spikes, drops, or outliers. Data points that are identified as anomalous based on their deviation from normal behaviour are labelled as "abnormal data points", usually referred to as outliers.

In supervised learning, data labelling assigns each data point a classification. Labelled data is used to train the predictive model so it can learn the difference between normal and abnormal behaviour. In unsupervised learning, where labels are not provided, the model identifies deviations from normal patterns without pre-classified data. Our objective is to derive predictive patterns that contribute to the emergence of anomalies, which are used for predicting the anomalies. To analyse the training dataset, we select data clustering or classification techniques based on the characteristics of the dataset. This is to identify various data labelling in the dataset and with the aid of domain expert analysis, we group the various data points into three data labelling as normal data points, precursor data points and abnormal data points. Correctly labelling precursor data points is essential for enhancing the effectiveness of our anomaly prediction framework. It leads to improved accuracy, early detection, and a better understanding of anoma-

lies, ultimately enabling more effective responses to potential anomalies and optimising their operations.

Once the patterns are identified, the next phase involves converting them into Linear Temporal Logic (LTL) formula representations, which are used as security properties in the runtime checker to implement anomaly mitigation strategies. Utilizing "precursor data points" as states, we selectively extract the sequence of states as traces and label them as "positive traces". Simultaneously, "abnormal data points" labelled as "negative traces," are also captured. Both traces (positive and negative traces) are converted into a binary representation, combined into an input file, and then processed by the LTL learning algorithm to produce an LTL formula. The formula is created by combining the identified temporal patterns and temporal logic expressions. The LTL formula generator uses temporal logic operators to express relationships and conditions over time. The LTL formula algorithm learns the formula from only positive traces, focusing on the behaviour indicative of potential anomalies in the system.

We evaluate the generated LTL by applying it to the testing data to make predictions based on the temporal patterns encoded in the formula. In runtime verification, the LTL formula is **implemented** as a system security property to determine whether the data violates specific properties. Each data point in the testing data is treated as a state, while a trace is a sequence of states. In runtime verification, a trace is a sequence or ordered set of events/states that represents the observed behaviour of a system over a period of time [7]. With LTL-based runtime verification, real-time monitoring of system operations is achieved while enforcing compliance with security properties defined using LTL to forecast and prevent such events.

The paper offers a dual contribution: firstly, it introduces a step-by-step process for generating system-specific LTL formulas through machine learning and applies them in the development of a runtime verification cybersecurity framework for cyber-physical systems. The second contribution lies in its ability to predict anomalies and trigger alarms using the system-specific LTL formula generated as a security property. This framework is applicable for tackling engineering/network data-related anomalies where temporal patterns can be identified within time series. Performance metrics, such as response time and throughput, are commonly used to evaluate runtime efficiency in CPS. This paper excludes the evaluation of runtime monitoring efficiency due to the unavailability of real-time data; instead, simulated datasets were utilized for experimental assessment.

The remainder of the paper is structured as follows; Section 2 reviews related research works, Section 3 is the preliminaries of LTL runtime verification, and Section 4 discusses our proposed LTL-based runtime verification framework and its systematic approach. In Section 5, we present our experimental evaluation through three case studies, while Section 6 discusses the experimental results from these case studies and provides a comparison with other related models. Finally, the conclusion and future research prospects are outlined in Section 7.

2. Related Work

This section explores the advancements in prediction-based anomaly detection and anomaly prediction. Additionally, it highlights the research gap in this domain and discusses how our framework addresses it.

2.1. Prediction Based Anomaly Detection

Anomaly detection focuses on identifying instances or patterns in data that deviate significantly from the norm or expected behaviour. This has been investigated for several decades, and one of the anomaly detection methods is based on prediction. The method involves training the predictor to learn the normal behaviour of a time series and using the prediction errors to identify anomalies in test sequences.

Malhotra et al. [28] presented a prediction-based anomaly detection method employing stacked long-short-term memory (LSTM) networks for anomaly and fault detection in time series data. The process involves training a network on non-anomalous data and using it as a predictor over multiple time steps. The resulting prediction errors are modelled as a multivariate Gaussian distribution to assess the likelihood of abnormal behaviour. However, Li et al. [27] identified a weakness in using only LSTM for prediction, noting that many predictors struggle to accurately predict values around abrupt changes in time series, which can result in false detection or missed detection.

Li et al. [27] presented anomaly scoring for prediction-based anomaly detection in time series. The paper attempted to address the issue of predictors failing to predict values around abrupt changes in time series accurately and proposed a framework using anomaly scoring for prediction-based detection. This framework employs Long Short-Term Memory (LSTM) networks for prediction and dynamic thresholding for anomaly detection from prediction errors. A clustering algorithm learns false positive (FPS) patterns from

training data, and test anomalies are scored based on their distance from these FPS patterns. The method is designed for anomaly detection but is not suitable for predicting anomalies.

Detecting only current or historical anomalies poses practical limitations [46], as the negative impact of an anomaly cannot be **prevented** once it has already occurred. While a predictor was employed in previous studies [28, 27], it was primarily used as a baseline for detecting anomalies rather than for forecasting them. Predicting the anomaly in advance can prevent its negative impact on cyber-physical systems. Anomaly prediction involves identifying and forecasting irregular patterns in data that deviate from normal behaviour. Early prediction of such anomalies aids in preventing potential issues or failures before they materialize.

2.2. Anomaly Prediction

Gu and Wang [18] presented a stream-based mining algorithm for online anomaly prediction to raise early alerts for impending system anomalies and suggest possible anomaly causes. The algorithm integrated Markov models with Bayesian classification methods for prediction, implemented within the IBM System S distributed stream processing cluster. Bayesian classification captured various anomaly symptoms to infer causes, while Markov models tracked changing patterns in different measurement metrics. The algorithm was claimed to predict and diagnose several bottleneck anomalies with high accuracy while imposing low overhead on the cluster system. However, the approach proposed by Gu and Wang [18] may not be suitable for predicting anomalies in Cyber-Physical Systems (CPS) at runtime. CPS consists of tightly integrated physical processes, such as temperature control, robotic movement, power distribution, chemical reactions, traffic flow in smart transportation systems, fluid dynamics, and structural health monitoring, alongside computational elements like sensors, control systems, actuators, data processing units, communication networks, software algorithms, and user interfaces. The integration of Markov models with Bayesian classification methods may not be directly applicable to CPS due to their complexity, real-time requirements, and large-scale, dynamic nature. Furthermore, CPS often involve noisy or incomplete sensor data, which can compromise the accuracy of these methods. Addressing the unique challenges of CPS requires specialized models that effectively account for the interactions between physical processes and computational elements, extending beyond traditional prediction approaches like Markov and Bayesian models.

In another work, Tan et al. [40] introduced ALERT, an adaptive runtime anomaly prediction system designed to improve prediction accuracy in dynamic hosting infrastructures such as IBM System S stream processing clusters and PlanetLab. ALERT uses a self-evolving learning algorithm to adapt to these environments, employing a triple-state multivariate stream classification scheme that includes an alert state, in addition to normal and anomaly states. This alert state represents measurements preceding an anomaly, enabling the model to capture pre-anomaly symptoms. For runtime anomaly prediction, monitoring sensors are deployed on all hosts within the infrastructure to continuously track metrics such as CPU consumption, memory usage, input/output data rate, and buffer queue length for each host and application component. However, the method proposed by Tan et al. [40] faces similar limitations as Gu and Wang [18]'s approach, making it unsuitable for predicting anomalies in Cyber-Physical Systems (CPS) at runtime.

In a related study, Langone et al. [25] identified model interpretability and explainability as key challenges for data-driven algorithms in industrial settings and proposed the Interpretable Anomaly Prediction (IAP) method for Industry 4.0. The methodology presented in the paper predicts anomalies by assessing whether future behaviour is anomalous. This is achieved by providing a probability, referred to as the anomaly score, which indicates the likelihood of future data being abnormal. The method consists of three steps: (i) data preparation, (ii) learning, and (iii) model refinement. This involves cleaning, transforming, and selecting features to standardize data and address the class imbalance, with regularized logistic regression at its core for interpretability.

El-Hadad et al. [16] proposed predicting abnormal electricity consumption using the isolation forest algorithm to classify smart meter readings as normal or abnormal. Based on these labels, Random Forest and Decision Tree algorithms forecast anomalies. Experiments revealed both methods could detect and predict anomalies 30 minutes in advance with comparable performance. While effective for handling dynamic power consumption data through auto-labelling, further research is needed to adapt this method for improved accuracy and robustness in diverse engineering environments.

To address the research gap in anomaly prediction for CPS, we present a framework that predicts anomalies using LTL-based runtime verification monitoring. Our framework integrates LTL-based runtime verification with data analysis to enable real-time monitoring, timely anomaly prediction, and computational efficiency. The primary advantage of our proposed framework

is its ability to provide formal guarantees, which are crucial for safety-critical applications. By utilizing Linear Temporal Logic (LTL) as a formal method, the framework ensures these guarantees while maintaining low computational requirements. Additionally, it facilitates the ability to respond effectively to anomaly predictions. Historical data is split into training and testing sets, with the training data used to identify temporal patterns leading to anomalies. These patterns are formalized into LTL formulas by combining indicative variables with temporal logic expressions that capture conditions and relationships over time. The generated formulas are then applied to test data to predict anomalies, even in cases with abrupt time-series changes.

According to Zheng et al. [49], runtime verification is an effective method for detecting subtle errors that may be difficult to identify due to scalability challenges, such as state explosion or unexpected interactions with the physical environment. Our framework leverages this advantage to detect errors that traditional methods might fail to recognise in CPS. Furthermore, as noted by Bernstein and Harter Jr [10], temporal logic has long been used to specify concurrent and real-time programs, serving as the foundation for the framework's correctness properties. To enhance predictive accuracy and adaptability, machine learning algorithms are selected through experimentation with specific datasets.

3. Preliminaries

The anomaly prediction framework is designed to anticipate anomalies before they occur. It leverages LTL as the foundational and most prevalent variant of temporal logic—a key family of specification languages in runtime verification [30]. This framework is built upon LTL formulas derived from historical CPS data, which are defined as system properties for runtime verification, which requires that correctness properties be articulated in a formal specification language. Runtime verification dynamically analyses and monitors a system's behaviour during execution to ensure compliance with specified properties or requirements [17]. As elucidated by Havelund and Peled [20], runtime verification can occur either online, as the system executes, or offline, by processing log files generated by the system. In online processing, runtime verification gathers real-time execution data and must promptly deliver a verdict, even without access to the complete sequence [20]. The real-time execution data are represented as time series and inputted as traces.

In this section, we present data formalization in Section 3.1 which is implemented in Section 4.3 and introduces LTL, the specification language for runtime verification, as detailed in Section 4.4 and runtime verification specification used in Section 4.5.

3.1. Data Formalization

This section formalizes key aspects of the data representation, including time series data, the relationship between time series, data points, states, traces, and the labelling process.

3.1.1. Time Series Data

A **data point** d_i represents an observation or measurement recorded at a specific time step t_i. In a dataset stored as a CSV file, each row corresponds to a data point, with columns representing the m attributes or features of the observation. In mathematical notation, $d_i \in \mathbb{R}^m$, where m is the number of attributes or features associated with each data point at each time step.
A **time series** is a sequence of time steps $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$, where t_i denotes the i-th time step. At each t_i, the corresponding data point d_i is observed, forming a time series $D = \{d_1, d_2, \ldots, d_n\}$. In a CSV file, the first column typically represents the time steps $\mathcal{T}$, while the remaining columns contain the features of each d_i. The time series can be expressed as:

$$D = \{(t_1, d_1), (t_2, d_2), \ldots, (t_n, d_n)\}.$$

A **state** s_i represents the condition of the system at a specific time step t_i. The state is derived from the data point d_i using a function f, such that:

$$s_i = f(d_i).$$

A **trace** τ is a sequence of states over time, capturing the evolution of the system through different conditions. Formally, a trace is defined as:

$$\tau = \{s_1, s_2, \ldots, s_n\},$$

where each state s_i corresponds to the time step t_i in the time series.
To illustrate the relationship between time series, data points, states, and

traces, consider the following structure of a CSV file:

Time t_i	Temperature	Pressure	Humidity
00 : 00	25.3	1012.5	60
01 : 00	24.8	1012.8	62
02 : 00	24.5	1013.0	63

For this dataset, the trace τ would be a sequence of states corresponding to the system's evolution over time.

- At $t_1 = 00:00$, the data point $d_1 = [25.3, 1012.5, 60]$ gives the state s_1.
- At $t_2 = 01:00$, the data point $d_2 = [24.8, 1012.8, 62]$ gives the state s_2.
- At $t_3 = 02:00$, the data point $d_3 = [24.5, 1013.0, 63]$ gives the state s_3.

The **trace** τ is thus:

$$\tau = \{s_1, s_2, s_3\} = \{[25.3, 1012.5, 60], [24.8, 1012.8, 62], [24.5, 1013.0, 63]\}$$

This trace represents the system's evolution over time, capturing the states at each time step in the sequence.

3.1.2. Labelling Process

Let $\mathcal{L} = \{l_1, l_2, ..., l_n\}$ represent the labels associated with each data point d_i in the time series D. Each label $l_i \in \{\text{Normal}, \text{Predictive}, \text{Anomalous}\}$ indicates the classification of the corresponding data point d_i.

The labelling process involves assigning a label l_i to each data point d_i based on whether:

- l_i = Normal: The data point d_i corresponds to expected or typical behaviour of the system.
- l_i = Predictive: The data point d_i serves as a precursor to an anomaly and indicates a potential future deviation.
- l_i = Anomalous: The data point d_i represents a significant deviation from normal behaviour, indicating an anomaly.

The set of labels $\mathcal{L}$ forms the labelled time series dataset:

$$\mathcal{D}_L = \{(t_1, d_1, l_1), (t_2, d_2, l_2), ..., (t_n, d_n, l_n)\}$$

This process extracts predictive patterns in Phase III (Section 4.3).

3.2. Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL), as a type of temporal logic formula, serves as a specification language for finite and infinite sequences of states referred to as traces. LTL operates from a linear-time perspective, where each point in time has a unique successor [30, 31]. LTL is widely used in various fields, including program verification, robotic motion planning, process mining, and more. It enables the specification of properties such as safety, fairness, and liveness.

Definition 3.1 (Syntax of LTL). *The syntax of LTL is given in the Backus–Naur form (BNF) below.*

$$F ::= \top \mid p \mid \neg F \mid F \wedge F \mid F \vee F \mid \mathbb{X}\, F \mid F\ \mathbb{U}\ F$$

In the above syntax, 'p' represents atomic proposition and there are two temporal modalities presented: $\mathbb{X}$ is written prefix and means *next*, and $\mathbb{U}$ is written infix and means *until*. The other modalities like F (future), G (globally), R (release) in LTL can be defined in terms of $\mathbb{X}$ and $\mathbb{U}$. Specifically $G\psi \equiv \bot R\psi \equiv \neg F \neg\psi$; $F\psi \equiv \top\ \mathbb{U}\ \psi$ and $\phi R\psi \equiv \neg(\neg\phi\ \mathbb{U}\ \neg\psi)$. In formal logic and temporal logic, ψ (psi) and ϕ (phi) are commonly used as placeholders for specific propositions, conditions, or properties to analyse in a system. They are general symbols that represent logical statements about the system's state or behaviour, allowing for the construction of complex expressions to describe how a system should behave over time.

The formal semantics of LTL are often defined using a Kripke model M, which is a tuple $M = (S, I, R, L)$ where S is a finite set of states, $I \subseteq S$ is a set of initial states, $R \subseteq S \times S$ is the transition relation between states and $L : S \to 2^{AP}$ is a labelling function that maps a state to a subset of atomic propositions. Here, AP is the set of all atomic propositions.

A (possibly infinite) sequence of states is called a *path*, which is denoted by w. Assuming that the indexing starts with 0, we write $w[i]$ for the $(i+1)$th state in w, and we write w^i for the sub-path starting from the $(i+1)$th state.

Definition 3.2 (Semantics of LTL). *The Kripke-style semantics of LTL is defined via a* forcing *relation* $\models$ *as follow:*

$M, w \models \top$ if and only if it is always true; $M, w \models p$ if and only if $p \in L(w[0])$; $M, w \models \neg A$ if and only if $M, w \not\models A$; $M, w \models A \wedge B$ if and only if $M, w \models A$ and $M, w \models B$; $M, w \models \mathbb{X}\, A$ if and only if $M, w^1 \models A$; $M, w \models A \,\mathbb{U}\, B$ if and only if there exists $i \geq 0$ such that $M, w^i \models B$, and for all $0 \leq j \leq i$, $M, w^j \models A$.

We say that the pair (M, w) of a model M and a path w force an atomic proposition p, written as $M, w \models p$ when p belongs to the subset of atomic propositions in *first state* in w. The classical logic connectives are interpreted in the usual way. M, w force $\mathbb{X}\, A$ if and only if M and the path w^1 starting from the *next* state force A. In other words, A is true in the next state. M, w force $A \,\mathbb{U}\, B$ if B is true in the future, and before that, A must be true.

To derive the Linear Temporal Logic (LTL) formula, the 'samples2LTL' algorithm [29], an off-the-shelf tool, is employed as the LTL formula generator. Neider and Gavran [29] presents two algorithms to learn formulas in linear temporal logic (LTL) from examples. The first algorithm reduces the learning task to satisfiability problems in propositional Boolean logic and generates the smallest LTL formula consistent with the given data. The second algorithm combines the SAT-based learning approach with classical decision tree learning algorithms, allowing scalability to real-world scenarios with many examples but without guaranteeing minimal consistent LTL formulas. Neider and Gavran [29] provides a detailed explanation of an algorithm.

3.3. Runtime Verification Specification

Runtime verification specifications use formal methods to define system properties. A common approach is to use logical formalisms, such as temporal logic, to specify the desired properties of a system. An important aspect of formal specifications in runtime verification is using theorems to ensure that these specifications are correctly defined and can be effectively monitored [26]. For our framework, we implement the theorem: correctness of monitor synthesis.

Definition 3.3 (Correctness of Monitor Synthesis). *Given a formal specification φ defined in a temporal logic, there exists an automaton $\mathcal{A}_\varphi$ that can be synthesized to monitor the execution traces of a system. The automaton $\mathcal{A}_\varphi$ correctly identifies whether the system's execution satisfies or violates φ.*

The theorem ensures that the monitors synthesized from formal specifications can reliably detect violations of the specified properties during the system's execution, providing confidence in the correctness of the monitored system. For our runtime verification, we use the Process Analysis Toolkit (Process Analysis Toolkit (PAT)) as a model checker, designed to incorporate advanced techniques for analysing event-based compositional system models [38]. PAT is an off-the-shelf tool [45] supporting various model-checking capabilities, such as checking for deadlock freedom, divergence freedom, reachability, LTL properties with fairness assumptions, refinement, and probabilistic models.

In this paper, we leverage the Communicating Sequential Processes (CSP#) module, an extension of CSP tailored for runtime verification. CSP is a formal language used to describe interaction patterns in concurrent systems, making it suitable for modelling event sequences (traces) within PAT. Processes and their interactions are modelled as a series of events, enabling analysts to evaluate system correctness by observing allowed or disallowed traces.

Using CSP# in PAT, we define systems and verify temporal properties through the following steps:

- **Specification**: Define properties using LTL. For example, the formula

$$\varphi = \mathrm{G}(\mathrm{A} \rightarrow \mathrm{F}\ B)$$

  states, "Globally, if A occurs, B must eventually follow."

- **Monitor Synthesis**: Construct an automaton $\mathcal{A}_\varphi$ from the LTL specification.

- **Correctness**: Prove that $\mathcal{A}_\varphi$ is sound and complete.

This approach integrates formal specifications with automaton-based correctness checks, ensuring precise runtime verification. By formalizing and proving the correctness of monitor synthesis, runtime verification can provide strong guarantees about the system's adherence to specified properties. For our framework, we generated LTL formula from the historical data and defined it as a system property φ for runtime verification in runtime checker using PAT language. We construct the automaton from a dataset that involves creating a formal model representing the patterns or sequences ob-

served in the data. During runtime, the monitor observes states/events and transitions between states.

4. The Proposed Framework

This section presents an anomaly prediction framework for cyber-physical systems. The proposed framework integrates data analysis, runtime verification, and LTL formula learning to predict anomalies at runtime. It learns an LTL formula from the system's historical data, uses it for real-time verification, and predicts anomalies with predefined responses for the system to act upon. The process is shown in Figure 2 and is divided into five phases; Phase I Data (Pre-preprocessing), Phase II (Pattern Discovery), Phase III (Featuring Engineering), Phase IV (LTL Property Learning) and Phase V (Runtime Monitoring). All processes preceding phase V are completed before the execution of the phase.

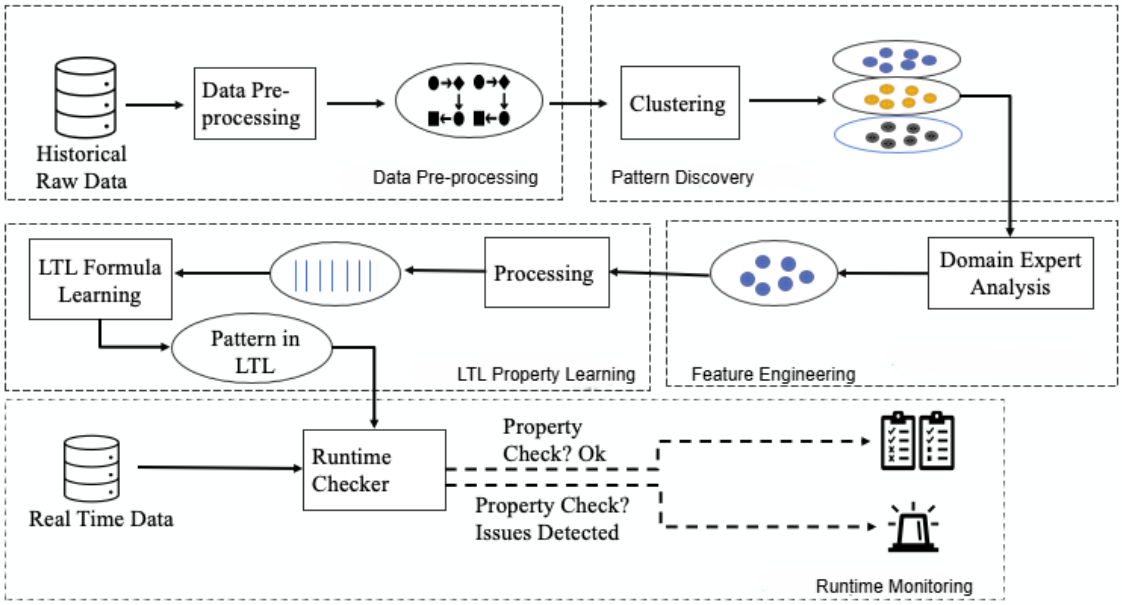


Figure 2: The proposed LTL based runtime verification framework for time series pattern detection.

Before delving into the various phases of our framework, we will consider the parameters for tuning in the phases.

Parameters

- **Data Split**: In Phase I (Section 4.1), we split our dataset into 70% training and 30% testing. The training set is used for model training, while the test set is held back for model evaluation and performance benchmarking.

- **Data Extraction**: In Phase III (Section 4.3), we extract a sequence of 'n' precursor data points, forming the predictive pattern P in time series. Additionally, a separate sequence of *'m'* anomaly data points are extracted to represent anomaly patterns and the extracted sequence excludes 'n'. These sequences of 'm' and 'n' data are stored in a trace file, separated by a delimiter, and used as input for the LTL algorithm to generate the formula of the sequence of 'n' precursor data points.

- **Trace**: In Phase V (Section 4.5), a sequence of data is chunked at intervals from the test data and fed into the runtime checker for verification. This chunk is represented by 's'. During the runtime verification phase, the test data is divided into 's' number of data, which are then fed into the runtime checker along with the LTL formula as the conformity property.

While we tuned our data to 70/30 data split, all other parameters tuning ('m', 'n' and 's') are dataset-specific for each case study. The choice of sizes *"m"* and *"n"* depends on the dataset and the characteristics of the attack event. We determined the optimal values through experimentation, testing various combinations and selecting the one that yielded the best results.

4.1. Phase I: Dataset Pre-processing

Dataset pre-processing is a critical phase in the data analysis and machine learning lifecycle. The primary goal of dataset pre-processing is to ensure that the data are accurate, complete, and relevant to the specific task at hand. The pre-processing of data includes data collection, data cleaning and transformation such as feature scaling, encoding categorical data, and data normalization. Other data processing based on data size includes data reduction (Dimensionality Reduction and Sampling).

In Phase I as indicated in Figure 2, our framework begins with data pre-processing. The dataset is expected to be numerical and time series variables with no missing values. Therefore, we begin the process with data collection,

followed by data cleaning and transformation. A dataset may contain various attributes like nominal, ordinal, binary, numeric, discrete, and continuous types. These attributes are pre-processed into a numerical format for compatibility with our technique. Depending on the dataset's characteristics and size, data feature selection, reduction, or encoding may be necessary. For our case studies, feature selection is used to reduce processing time and extract relevant features from large datasets. Systematic data reduction can be achieved through algorithms such as the Shapley value or Recursive Feature Elimination (RFE).

Also, during the data pre-processing, we perform data normalization, a process to scale numerical features within a dataset to a standard range to ensure that different variables in the dataset contribute equally to analyses. Another important step in data preprocessing is data binarization, which involves converting numerical variables into a binary format. Representing data variables in binary is essential for efficient computation and analysis. Additionally, the LTL algorithm requires input data to be in binary format.

We split the historical dataset into training and test data. The training data is processed to generate the LTL formula which serves as the property for runtime verification. The test data is used to evaluate the performance of the runtime verification.

4.2. Phase II: Pattern Discovery

The subsequent phase of our framework is pattern discovery, represented in Figure 2. This process involves identifying regularities or typical behaviours in data and recognizing deviations from these patterns as anomalies using a suitable machine learning algorithm that is determined through experimentation and evaluation on specific datasets. This process aims to achieve anomaly detection as its expected result.

The choice of algorithm depends on the task, the characteristics of the data, and the desired outcomes. During this stage, the machine learning algorithm is used in learning patterns or instances in data that deviate significantly from the norm. Machine learning has been applied in many fields to detect patterns useful for cybersecurity analysis [3, 4, 9, 43]. The approach may be either unsupervised learning or supervised learning (if labelled data is available).

In unsupervised learning, unlabelled data is given to a machine learning algorithm as input to find patterns, structures, or relationships within the dataset. Clustering is a common task in unsupervised learning. A clustering

algorithm is an artificial intelligence process that 'learns', that is, leverages data for improvement of performance on some set of tasks [14, 24]. Examples of clustering algorithms include K-means clustering, spectral clustering, agglomerative clustering, mean shift, OPTICS, hierarchical clustering, and DBSCAN (Density-Based Spatial Clustering of Applications with Noise). Clusters are formed in such a way that data points within the same cluster are similar, and data points in different clusters are dissimilar.

In contrast, supervised learning relies on labelled data to identify patterns to predict outcomes in new unseen data. The dataset includes input features paired with correct output labels, serving as the training set for algorithms like K-Nearest Neighbors (KNN), Random Forest, Support Vector Machines (SVM), Linear Discriminant Analysis (LDA), and Neural Networks.

In both supervised and unsupervised learning, inaccurate results can lead to suboptimal system performance, a failure to achieve the intended objectives, and overall effectiveness can be compromised. Data misclassification can lead to misinformed decision-making and can pose security risks. Failing to identify malicious activities may compromise the security of systems and sensitive data. Determining the ideal number of clusters or classes in clustering or classification is a crucial factor that contributes significantly to achieving precise results. Techniques such as the elbow method [42], silhouette score [36], or cross-validation [34] can be employed to determine the optimal number of clusters or classes. To determine the best outcome, classification algorithms are evaluated using metrics like accuracy, precision, recall, and F1-score, while clustering algorithms are assessed with the Silhouette Score. Aside from using the systematic approach, domain expert analysis can be used in the selection of data classification for security analysis.

4.3. Phase III: Feature Engineering

The subsequent stage in our framework is feature engineering, as illustrated in Figure 2, which involves domain knowledge expert analysis. Feature engineering is the process of using domain knowledge to select, modify, or create new features (variables) from raw data that can improve the performance of machine learning models. In this phase, the expected result is to predict anomalies by identifying and extracting patterns that contribute to the occurrence of anomalies identified in phase II (section 4.2). Integrating domain-specific knowledge is crucial for identifying significant features and clusters in cyber security analysis, reflecting anomalies. This phase comprises two stages: first, selecting a group or cluster that represents each category

of data point, and lastly, an extraction process of traces that are "precursor data points" and "abnormal data points".

4.3.1. Group selection

While machine learning identifies patterns in historical data by grouping similar points, domain-knowledge expert analysis helps to pinpoint which group or cluster represents a "normal data point", "precursor data point" and "abnormal data point".
A "normal data point" refers to a data observation that conforms to the expected or typical behaviour within a dataset. In data analysis, normal data points are those that do not exhibit unusual patterns, deviations, or anomalies. For "precursor data point", it refers to a data point that precedes or leads to an anomaly event in a sequence of data. It indicates the data point that serves as a precursor to the anomaly event, providing information or context that may be relevant to understanding or predicting the anomaly. The "abnormal data point" refers to a data observation that deviates significantly from the expected or typical behaviour within a dataset. In data analysis, abnormal data points are those that exhibit unusual patterns, deviations, or anomalies compared to the majority of the data.

The various data points may be dynamically identified using clustering techniques if clear distance metrics distinguish abnormal, precursor, and normal data points. In the case where the algorithm is unable to identify the precursor data points, domain knowledge experts may be able to identify precursor and abnormal data points. This identification is crucial for subsequently extracting patterns which are later translated into Linear Temporal Logic (LTL) formulas. The LTL formula is generated based on the set of precursor data points as positive traces and abnormal data points as negative traces.

4.3.2. Extraction process

With the help of domain knowledge, we establish clusters or instances that categorize normal, precursor, and anomaly patterns. Using insights from the group selection, we create a trace file as input to generate the LTL formula. As discussed in Section 4, n denotes the number of states of precursor data points. For the anomaly instance, m represents the number of states of the abnormal data points, the choice of m must exclude the data sequence of n to ensure that only normal instances are extracted. The selection of n and m depends on the behaviour of anomalies and their occurrence in the time

series, as determined by domain expert analysis. Incorrect selection may affect the accuracy of the temporal behaviour. In data analysis, a row of data represents the system's state at a specific point in time. Thus, in our discussion, the terms "row" and "state" are used interchangeably to convey the same concept. The group selection and extraction processes are carried out using the Algorithm 1:

Algorithm 1 processes a dataset from a CSV file by extracting patterns based on the values in the 'Cluster_Label' column, which serves as the cluster identifier. The 'Cluster_Label' is a categorical variable that assigns each data point to a specific cluster or category based on its similarity to other data points, typically determined by clustering or classification algorithms. Algorithm 1 organizes and extracts the data according to these cluster identifiers to generate *predictive* and *anomaly* traces. These traces are then combined into a single trace file for further analysis. This output trace file will serve as the input for the next phase, LTL formula learning.

4.4. Phase IV: LTL Property Learning

The next phase of our framework is the LTL formula learning (Phase IV according to Figure 2). During this phase, we learn the LTL formula derived from the outcome of phase III (section 4.3). In the process of learning Linear Temporal Logic (LTL) formulas from historical datasets, it is anticipated that the dataset encompasses both regular and anomalous events. The LTL formula is influenced by the temporary patterns observed in the dataset and the interplay among the data variables if it exists. This phase aims to learn the LTL formula that distinguishes positive (P) and negative (N) trace sets, where the formula should serve as a property for every trace in P and not for any trace in N.

To derive the LTL formula, we employ samples2LTL algorithm [29], a technique used to automatically generate LTL formulas from execution traces. According to Neider and Gavran [29], the repository offers two methods: one encodes the problem as the satisfiability of a Boolean formula, providing it to the Z3 solver, while the other is grounded in decision tree learning. The concept involves transforming the observed behaviour of a system, represented as a trace (τ), into a formal LTL formula that captures specific desired properties or behaviours. The trace τ is formalized in Section 3.1.1 and defined as a sequence of states or events. The algorithm outputs an LTL formula capturing the observed properties or behaviours in the given trace. A set of atomic propositions is defined that describe properties or conditions in the

Algorithm 1 Data Extraction and Trace Compilation

Load CSV file into DataFrame ‘df‘
if ‘Cluster_Label’ column does not exist in ‘df‘ **then**
  Print "Error: ‘Cluster_Label’ not found."
  return
end if
Initialize lists: normal_patterns ‘N’, anomaly_patterns ‘A’, predictive_patterns ‘P’
Count unique clusters in ‘Cluster_Label’
if clusters 0, 1, and 2 exist **then**
  for each row in ‘df‘ **do**
    if ‘Cluster_Label’ == 0 **then**
      Extract m rows as ‘N’: $N = \{d_t, \ldots, d_{t+m-1}\}$
    else if ‘Cluster_Label’ == 1 **then**
      Extract m rows as ‘A’: $A = \{d_{t'}, \ldots, d_{t'+m-1}\}$
    else if ‘Cluster_Label’ == 2 **then**
      Extract n rows as ‘P’: $P = \{d_{t''}, \ldots, d_{t''+n-1}\}$
    end if
  end for
else if clusters 0 and 1 exist **then**
  for each row in ‘df‘ **do**
    if ‘Cluster_Label’ == 0 **then**
      Extract m rows as ‘N’
    else if ‘Cluster_Label’ == 1 **then**
      Extract m rows as ‘A’
    end if
  end for
  Extract n rows leading to ‘Cluster_Label’ == 1 as ‘P’
end if
Establish t (predictive) and t' (anomaly, $t' > t$)
Compile positive trace from ‘P’ and negative trace from ‘A’
Write traces to ‘patterns.trace’ with separator “—”

system and are derived from the states or events in the trace. For instance, a trace like $1, 0, 1; 0, 0, 0; 0, 1, 1$ defines three states, each indicating values for three propositions x_0, x_1, x_2. The Samples2LTL algorithm analyses the given trace and converts it into an LTL formula based on the observed behaviours.

Using the algorithm, an LTL formula is generated with the combination of atomic propositions and temporal operators as identified in Section 3.

In our framework, the Samples2LTL algorithm takes the trace file derived from Phase III (section 4.3) and generates LTL formula. LTL formulas, which serve as a formal language for expressing temporal properties, play a crucial role in runtime verification, dynamic monitoring, and assessing system behaviour during execution.

4.5. Phase V: Runtime Monitoring

As shown in Figure 2, Phase V (Runtime Verification) represents the final stage of our framework, which involves runtime verification. In this phase, we verify the conformity of the system model with the property specification defined using the derived LTL formula in phase IV (section 4.4). At this stage, the runtime checker assesses real-time data against temporal properties for ongoing monitoring. The generated temporal properties are employed to predict the likelihood of an anomaly in the system in real-time. When an anomaly is predicted, the runtime verifier immediately triggers an alarm to alert the system in advance, enabling proactive interventions before the anomaly occurs.

Hou et al. [22] introduced a runtime verification engine for the digital twin, which is capable of verifying properties expressed in various temporal logic languages. The paper [22] highlighted that runtime verification utilizes some variants of LTL, including LTL on finite traces (LTLf) and three-valued LTL (LTL_3). The Process Analysis Toolkit (PAT), a model checker, supports both LTLf and PTLTL (Past-time LTL), with PTLTL being another useful language for specifying security-related properties [15]. In this paper, we implement the runtime verification engine with the declaration of the LTL formula as the property. This paper adopts LTL on finite trace (LTLf) with strong next, that is, $\mathbb{X}\ A$ is true when the next state exists and makes A true; otherwise, $\mathbb{X}\ A$ is false. In this scenario, $\mathbb{F}\ A$ is only true when there is a future state that makes A true; otherwise, it is false.

In our framework, we integrate runtime validation using an algorithm based on the runtime-monitor script from Hou et al. [22]'s work. Runtime monitoring involves three phases: runtime modelling, property definition, and runtime checker.

Runtime Modelling:. We construct the automaton. A system can be modelled using real-time data captured from the system as input. In our frame-

work, we are mainly interested in verifying properties over the state variables of the system. Let us name the state variables as *var1, var2,* $\cdots$, a state S is simply a snapshot of the values of state variables, i.e.,$S::=$ {*var1 = val1, var2 = val2,* $\cdots$ }. In PAT, we model a state via a process in Communicating Sequential Processes [21] with C# (CSP#) [39]. The process performs variable assignments as below.

$$S() = \{svar1 = val1; svar2 = val2; ...\} \rightarrow Skip;$$

The final trace T is a sequence of states, modelled as follows.

$$T() = S1(); S2(); ...$$

Property Definition:. The user can define properties over state variables. For example, the following code defines a proposition that states "var1 is not 0". Linear Temporal Logic (LTL) is used for specifying properties of the system's behaviour. LTL is based on temporal operators that describe how properties evolve.

Runtime checker:. The generated LTL formulas are incorporated into a runtime checker. The runtime checker dynamically monitors the system's execution and evaluates it against the specified LTL formulas. During the execution of the system, the runtime checker continuously observes its behaviour in real-time. It checks whether the temporal properties specified by the LTL formulas are satisfied at different points in the execution.

The foundation of our runtime verification framework is based on the observation that verifying LTL with finite traces in Process Analysis Toolkit (PAT) language corresponds to verifying FLTL with strong next/future. It provides a means to detect and respond to violations of desired properties during system operation. In the PAT model checker, for instance, if the property condition is defined as:

$$\#define \;\; v1Safe \quad (var1! = 0);$$

We verify whether the observed behaviour, defined as a sequence of states (S) or Trace(), satisfies the specified LTL formula, v1Safe. This process involves evaluating the formula against the observed sequence of states S to determine if it holds true. Given the property stated above, using the PAT model checker, we check the safety property that "var1 should never be 0" using the temporal modality G, which is written as $\Box$:

$$\#assert \quad Trace() \models \Box v1Safe;$$

For our framework, the runtime verification outcome can be categorized into two possibilities:

- If the system's behaviour (S) conforms to the specified temporal properties, the verification result is positive (true). This implies that there is a tendency for violation or violation is taking place.
- If the system behaviour (S) does not conform to the specified temporal properties, the verification result is negative (false), which indicates that the system behaviour has no violation.

For our framework, the generated LTL formula from the historical data is used as a security property and is defined for runtime verification as the runtime checker in the PAT (CSP) language. In the case of a violation, the runtime verification tool may generate alerts or reports to notify relevant stakeholders about the detected deviation from the expected behaviour. These alerts can be used for further investigation or corrective actions. Depending on the severity of the violation and the nature of the system, runtime verification outcomes may trigger adjustment mechanisms, modification strategies, dynamic response techniques, or corrective measures to restore the system to a compliant state.

5. Experimental Evaluation

In this section, we discuss three case studies to demonstrate the implementation of our framework and evaluate its performance. Our experimental exercise begins with the simplest system dataset obtained from the CubeSat Communication system as the first case study and proceed to more complex system data analysis. For the CubeSat Communication system, we collated an unlabelled dataset acquired from a testbed designed to replicate a genuine CPS environment while the remaining two case studies which come with labels (the BATADAL team's work [41], and Power Substation Automation Systems [44]) were obtained online. In all the experiments, we used the parameters described in Section 4 and are tuned as necessary based on the nature of each data.

- Train-test Split: The engineering datasets were split into training and testing data in a 70-30% split ratio except for the Power Substation Automation Systems dataset where the training phase employed attack datasets associated with IED1, and the testing phase utilized attack datasets related to IED2.

- Framework Parameter: The parameters for each case study are presented in Table 1. Due to the lack of a real-time system, we utilized the testing data as the runtime data to create a system model and implemented our runtime model checker using the Process Analysis Tool (PAT).

Case Studies	**Data Extraction**		**Trace**
	n	**m**	**s**
Case 1: CubeSat	10	10	30
Case 2: BATADAL	20	15	50
Case 3: Power Substation Dataset	20	15	50

Table 1: This table depicts the parameters set for data processing for each case study.

For supervised learning, data are trained on labelled data while for unsupervised learning, data are trained on unlabelled data. In the unsupervised machine learning scenario of a CubeSat Communication dataset (an unlabelled dataset), clustering techniques are used to reveal patterns and structures within the data. The last two case studies employ supervised machine learning, using labelled columns for data classification to facilitate precise predictions of new data. The effectiveness of our prediction depends on the accurate clustering or classification of the data in a time series. Therefore, we evaluate the machine learning algorithms for clustering the unlabelled dataset to select the most suitable one using the Silhouette Coefficient, while assuming that the labelled dataset is correctly labelled. The one-hot encoding algorithm is used to convert numerical variables into binary representations. We evaluated the effectiveness of the clustering algorithm using clustering evaluation metrics such as the Silhouette Coefficient, a metric useful for unlabelled data evaluation. In addition, classification evaluation

metrics such as accuracy, precision, sensitivity, and the F1 score are used for supervised learning scenarios where the true labels are known.

- Evaluation assessment: In anomaly detection, evaluation centres on pinpointing the exact moment an anomaly occurs, with metrics primarily assessing the accuracy of identifying these specific time points. Conversely, anomaly prediction focuses on predicting anomalies before they happen within a time series. In each case, our evaluation is based on how early the prediction is made, ensuring it occurs before the anomaly period begins. We evaluate our framework using evaluation metrics to assess our prediction performance. The evaluation metrics, which include prediction time, precision, sensitivity, and F1-score, are defined as follows:

  - Prediction Time: This refers to the interval between when an anomaly prediction is made and when the actual anomaly occurs. In anomaly prediction, prediction time is critical, as it determines how much lead time is available for preventive actions. An effective prediction time allows for sufficient warning, enabling operators or systems to respond and potentially mitigate the impact of the predicted anomaly. In our framework, the prediction time is determined by the number of states leading up to the anomaly detection. Based on the system properties defined, the runtime checker verifies each trace fed into the system to ensure conformance. When a violation is detected, it typically triggers a predefined response process, such as an alarm, to handle potential issues. The immediate response helps prevent the anomaly from affecting the system's operations. This deviation signifies that the system's behaviour has deviated from the expected norm, which could indicate an anomaly, fault, or unexpected condition.
  - Precision: It is also known as Positive Predictive Value, which measures the proportion of correctly identified positive instances (true positives) out of all instances predicted as positive. Precision in anomaly prediction assesses how often predictions of future anomalies are correct.
  - Sensitivity: It is also known as Recall or True Positive Rate, measures the proportion of actual positive instances that were cor-

rectly identified. Sensitivity measures how well the model identifies anomalies in advance

- F1 score: The F1 Score is the harmonic mean of Precision and Recall, providing a balanced measure that considers both false positives and false negatives. It is a robust measure when both avoiding false alarms and capturing true anomalies are crucial.

The result analysis is presented in each case study, while the discussion of the outcomes can be found in Section 6. The test data is divided into traces based on the parameters specified in Table 1 to perform the evaluation. Runtime verification is conducted using a runtime checker that examines traces to determine whether the system's behaviour aligns with predefined properties, specifically the LTL formulas derived for each case study. We use PAT as a runtime checker and by continuously comparing observed behaviour with these properties, runtime verification swiftly detects deviations in advance of an anomaly event in time series. When a deviation is detected by the runtime checker, it typically triggers a predefined response process designed to handle potential issues. This deviation indicates that the system's behaviour has been violated, which could signify an anomaly, fault, or unexpected condition.

5.1. Case Study 1: Satellite Communication Systems

This case study examines a CubeSat experimental setup utilizing Processor-in-the-Loop (PiL) testing to simulate a satellite communication system. The setup consists of two main components: the ground station (GS) and the satellite itself. The ground station includes equipment for transmitting, receiving, and other necessary functions. The satellite, which is a physical twin of the actual satellite, communicates with the ground station through uplink and downlink channels. LabView, a software development environment, was used to configure different parameters and establish communication between the ground station and the satellite.

An injection of false data was performed and the data were captured in text format. This led to two different attacks on the CubeSat system which are GPS spoofing and attitude control manipulation. For the GPS spoofing attack, a false GPS signal was sent to mislead the satellite's navigation and positioning systems. An attacker sends fraudulent GPS signals that mimic legitimate signals from GPS satellites. These false signals interfered with

the genuine GPS signals received by CubeSat. The CubeSat's onboard GPS receiver processes the spoofed signals, causing it to calculate an incorrect position and trajectory. This misled the satellite about its location and movement. The CubeSat responded to the manipulated position data by making erroneous adjustments to its orbit or attitude, resulting in the second attack. In Attack 2 (Attitude Control Manipulation), the attack targeted the satellite's attitude control system, which is responsible for maintaining the satellite's orientation in space. By manipulating the system, the attacker disrupted the satellite's ability to properly align with its intended position, affecting its precision in navigation and orientation. This interference led to issues with the satellite's positioning systems, ultimately hindering its ability to carry out its designated functions, such as communication, imaging, or data collection.
During the CubeSat experiment, engineering data such as the ground station processor temperature, satellite processor temperature, the ground station processor voltage, satellite processor voltage, rotational speed, and satellite magnetic angle, are captured. Various parameters at normal operations are shown in Figure 3.

Engineering data were collected under two conditions: normal and attack/abnormal. The unlabelled data collection lasted for 180 seconds. For traffic captured under normal conditions, the temperature processor readings were captured under an average room temperature of about 24°C while the processor voltage was about 3V for both the ground station and satellite station. For the rotational speed, a value of 0 was captured as normal from the satellite while 163° was observed as the normal magnetic angle of the satellite. Likewise, while the experiment was carried out in the laboratory, the conditions were varied to show an attack scenario. The impact resulted in a change in processor voltage, temperature, magnetic angle, and rotational speed. The captured data was not labelled and an unsupervised machine learning technique will be required to analyse the captured data. Shown in Figure 3 is the visual of the statistical distribution of the dataset obtained from the CubeSat experimental setup.

Data Pre-processing. The collated engineering data is pre-processed by cleaning the raw data and translating it into a format useful for machine learning analysis. Data normalization is part of the pre-processing. The data normalization process involves transforming variables to a common scale. The numerical variables are converted into binary representations using the K-

means Clustering algorithm.

Pattern Discovery. The next step in the LTL based runtime verification framework is to learn temporary patterns. Given the type of engineering data obtained, which is an unlabelled dataset, we implement an unsupervised learning approach. The step towards learning temporary patterns is the selection of a suitable clustering algorithm and choosing the number of clusters (k). The clustering outcome is assessed using the Silhouette Coefficient, a suitable evaluation metric for unsupervised learning. Using the elbow method, we identify the optimal number of clusters (3) for various unsupervised learning algorithms. Following data standardization, we apply the algorithms with K = 3 and evaluate their performance. Compared to the other algorithms in Table 2, the evaluation result highlights hierarchical clustering as the preferred option, achieving a score of 0.7365, which indicates well-defined and well-separated clusters.

Evaluation Metrics	**KMeans**	**Spectral**	**Hierarchical**	**GMM**	**BIRCH**	**Mean Shift**	**DBSCAN**	**AGNES**
Silhouette Coefficient	0.6652	0.7118	**0.8521**	0.3152	0.6051	0.6652	0.6652	0.6652

Table 2: The assessment of CubeSat dataset clustering through different algorithms points to hierarchical clustering as the favoured choice, with a Silhouette Coefficient value of 0.7365.

Clustering presents three clusters (0, 1, 2). With the aid of domain knowledge, a cluster indicative of the normal, predictive and anomaly are identified.

Featuring Engineering. By implementing Agglomerative hierarchical clustering, the dataset is clustered into three groups, cluster '2' is identified as the group associated with precursor data points, cluster '0' corresponds to the group of normal data points while Cluster '1' is identified as the group associated with abnormal data points.

By applying the Algorithm 1, we extracted n states of the precursor data points as predictive patterns and m states of the abnormal data points as anomaly patterns. The values of n and m are indicated in Table 1. The predictive patterns and anomaly patterns were extracted from positive and negative traces respectively. The respective traces are then stored in a "patterns.trace" file to be used as input into the LTL formula generator algorithm.

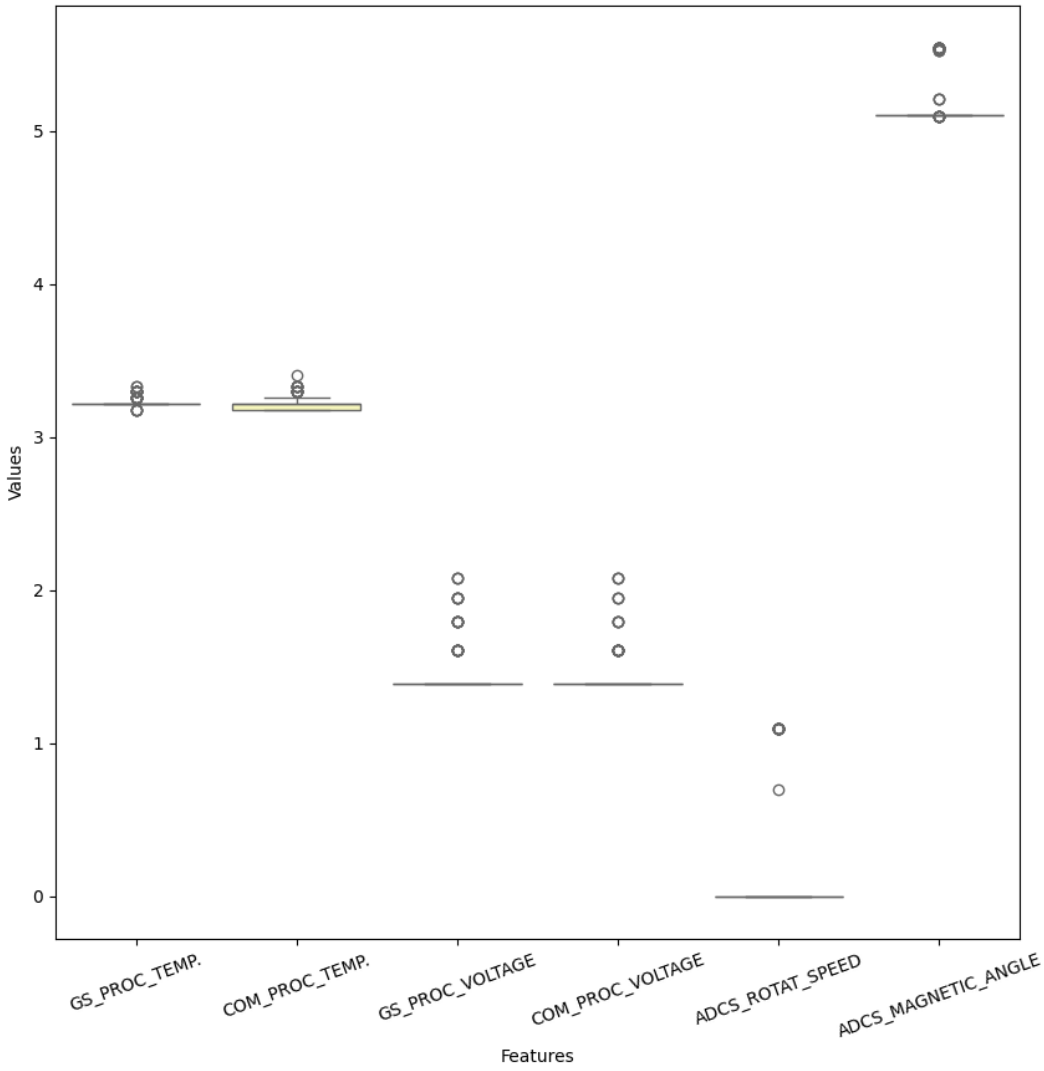


Figure 3: The graph displays the statistical distribution of data points for the CubeSat dataset, with the x-axis representing features and the y-axis showing the distribution of their values, which includes whiskers and outliers.

LTL Property Learning. In this phase, we learn the LTL formula to describe sequential patterns in data or system behaviour. After generating the trace

file, a representation of the temporary patterns learned from the training dataset, we fed the file into the Traces2LTL algorithm. As a result, it generated the Linear Temporal Logic (LTL) formula as shown in Table 3:

Proposition	Features	Normal Value Range (Inliner)	Formula
x1	GS TEMPERATURE	23-25	$\neg F(x_1)$
x2	SAT TEMPERATURE	23-25	$\neg F(x_2)$
x3	GS PROCESSOR VOLTAGE	3	$\neg F(x_3)$
x4	SAT PROCESSOR VOLTAGE	3	$\neg F(x_4)$
x5	SAT ROTATIONAL SPEED	0	$\neg F(x_5)$
x6	ADCS MAGNETIC ANGLE	163	$\neg F(x_6)$

Table 3: Table showing the Case Study 1 LTL propositions, dataset features, their normal value ranges (Inliner) representing normal states, and the corresponding LTL formulae.

The LTL formula encompassed propositions spanning from $x1$ to $x6$. This formulation emerged because there were no discernible connections between the parameters denoted by these propositions despite the acquisition of patterns from each parameter.

In Linear Temporal Logic (LTL), the expressions $\neg F(x_3)$ and $\neg F(x_6)$ indicate that, at a specific future time, the anomaly events at positions $x3$ and $x6$ will ultimately become true, where the propositions $x3$ and $x6$ correspond to "GS_PROCESSOR_VOLTAGE" and "ADCS_MAGNETIC_ANGLE" respectively within the dataset. Applying the LTL formula to a satellite communication system dataset implies that anomalies remained false in the time series until temporary patterns emerged, signalling a departure from normal system operation. Analysis revealed that a voltage of 3.19 volts and a magnetic angle of 163 degrees represented values associated with normal system operations. The LTL formula, applied to variables "GS_PROCESSOR_VOLTAGE" and "ADCS_MAGNETIC_ANGLE" (denoted as $x3$ and $x6$ respectively), asserts that an anomaly is expected in the future if there is a deviation from the normal value ranges, indicative of temporary patterns.

Runtime Monitoring. Following the acquisition of the LTL formula, we move on to the subsequent phase, runtime verification. A CubeSat communication

model is developed using a testing dataset that represents 30% of the total historical dataset. This testing dataset is partitioned into various traces, with each trace representing a sequence of states. These states are then fed into a runtime checker to assess the system's adherence to the property defined by the generated LTL formula. The runtime verification process, along with its associated property (the generated LTL formula), is scripted in PAT, as elaborated in Section 4.5.

The defined property aims to determine the occurrence of anomalies in any state within the sequence of states. The verifier outputs "true" if such a state is identified, or "false" if no anomalies are observed throughout the sequence of states.

5.1.1. Result Analysis

Evaluating our framework techniques involves a comprehensive analysis of their performance, accuracy, and efficiency in predicting anomalies within CubeSat communication systems. We identify the first instance of prediction in the time series and then apply other evaluation metrics, including precision, recall/sensitivity, and F1-score, as discussed earlier in the evaluation section, to assess the effectiveness of our techniques in accurately identifying anomalies.

In the first prediction before the anomaly, the framework made predictions 15 states before the occurrence of an anomaly. Two instances of attack events were observed in the testing data. The results indicated that two instances of predictive events were observed. A high precision value of 1.0 indicates that the model's predictions are mostly correct and signifies that the model accurately predicted most instances within the dataset. Likewise, 1.0 is a high recall or sensitivity value which indicates that the model is effective at capturing most of the positive instances. The F1 score, representing the harmonic mean of precision and recall, is 1.0 in this instance. For this case study, the evaluation results demonstrate that our framework is highly effective in predicting anomalies from the testing data. These results are further compared with other case studies in Table 7, highlighting the framework's accuracy in anomaly prediction.

5.2. Case Study 2: Water Distribution System

This is a case study of cyber attacks on the water distribution system of C-Town, a Supervisory control and data acquisition (SCADA) system and dataset was created and published by the BATADAL team [41]. Developed

for an international competition aimed at securing critical infrastructure, the dataset features time-series data from a simulated water distribution system. This data includes normal operations and scenarios where cyber-attacks manipulate sensor readings or control commands. C-Town consists of 388 nodes linked with 429 pipes and is divided into 5 district-metered areas (DMAs). The BATADAL data include the water level at all 7 tanks of the network (T1–T7), the status and flow of all 11 pumps (P_U1 to P_U11) and the one actuated valve (V2) of the network, and pressure at 24 pipes of the network that correspond to the inlet and outlet pressure of the pumps and the actuated valve.

The BATADAL dataset includes various types of cyber-attacks that simulate real-world strategies to disrupt water distribution. For instance, in Attack 1, pressure features P_U10 and P_U11 are manipulated from 0 to 1. Attack 2 involves replaying sensor data to create fluctuations in features like P_U1 and P_U2, which, although within normal ranges, can cause issues like overflow if sustained. Attack 3 is more complex, involving simultaneous manipulation of multiple features. As described in the study [41], three datasets (testing and two training datasets) were created. For our experiment, the Training2 dataset was used which spans approximately six months and features several attacks, some of which have been provisionally labelled. We expanded the size of the dataset with synthesized data. After collating the dataset, we implement our framework to learn a pattern from the dataset. Figure 4 is the visual of the dataset for the water distribution system dataset.

Data Pre-processing. The SCADA system monitors a comprehensive set of 43 variables, which encompass the water levels within the tanks (7 variables represented as L_), as well as the inlet and outlet pressure measurements for both the actuated valve and pumping stations (12 variables denoted as P_). Additionally, it includes the flow rates and operational statuses of the actuated valve and pumps, represented by 24 variables denoted as F_ and S_ respectively. We conducted a comprehensive data pre-processing, involving format conversion, and data normalization.

Pattern Discovery. With the 70% training data, data classification is performed using various classification algorithms, including the random forest algorithm as identified by Aghashahi et al. [2]. Using evaluation metrics and a confusion matrix, the result reveals that the random forest algorithm is more efficient for anomaly detection of the case study data set than other

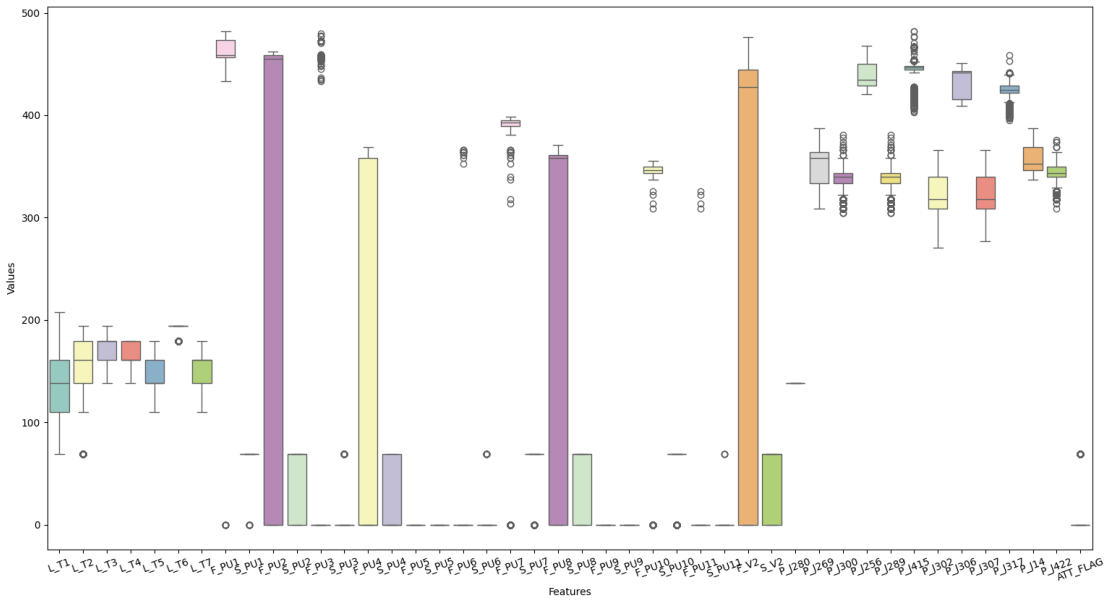


Figure 4: The graph indicates the statistical distribution of data points for the Water Distribution System dataset, with the x-axis representing feature names and the y-axis showing the distribution of their values

algorithms. The evaluation result of the random forest algorithm is as follows: True Positives (492), True Negatives (3685), false Positives (0), and false Negatives (0). Furthermore, the accuracy, precision, recall and the F1 score are equal to 1.

Featuring Engineering. With the aid of domain expert analysis, we also assessed whether the predicted label of the model derived from the classification algorithm aligns with the actual label. The dataset is attack labelled ("ATT_FLAG") with a 1/0 label column, with 1 representing the abnormal data point and 0 representing the normal data point.

The next step after confirming the anomaly detection accuracy is the extraction of temporal patterns leading to the anomaly event. We followed the pattern discovery process as explained in phase 4.3. The n precursor states leading to the abnormal data point were extracted, encoded using techniques like one-hot encoding, and saved as positive traces. Similarly,

m states 'before' and 'after' the abnormal point were encoded and stored as negative traces. The values of n and m for this case study are listed in Table 1, and the traces are saved in a trace file.

LTL Property Learning. The next phase is the definition of properties, expressed using Linear Temporal Logic (LTL). We acquire LTL formulas using the samples2LTL algorithm with the trace file as input to deduce the system property. In PAT, we define the LTL formula as a property in the *.csp* format. Table 4 presents the result of the generated LTL formula, and the proposition along with its representations are represented in the table.

The LTL formula is input as the security property which is used in the runtime checker.

Runtime Monitoring. We performed the runtime verification of the water distribution system by creating the runtime model of the water distribution system from the testing data and verifying the system's conformance to the derived LTL formulas. The testing dataset is chunked to form a series of traces with each trace containing six (6) states of data. Each trace is checked and input into a runtime checker to verify if the learned LTL formula set as the property of the system holds. The runtime verification discloses "true" if there is a state where the property holds, or "false" if there is no presence of anomalies at any state in the verified sequence of states.

5.2.1. Result Analysis

We evaluate our framework's performance based on prediction time, precision, sensitivity and F1-score. A predicted pattern is detected four states before the anomaly occurs. This deviation indicates that the system's behaviour has violated normal behaviour, which could signify an anomaly, fault, or unexpected condition. In our data analysis, there were a total of 35 instances of attacks from the synthesised data. Shown in Table 7 is the result of the analysis. In this case, a precision of 0.91 was recorded, indicating that the model correctly predicts a positive instance of approximately 91% of the time, with a sensitivity of 0.94 (94%) indicative of a high proportion of true positives and an F1 score of 0.93 suggests a good trade-off between precision and recall.

5.3. Case Study 3: Power Substation Automation Systems

This case study presents a simulated test environment for an IEC 61850-compliant substation automation system (SAS), as established by the re-

Proposition	Feature	Normal Value Range (Inliner)	LTL Formula
x2	L_T1	0 - 4	
x3	L_T2	0 - 5	!(F(x30));
x5	L_T4	2 - 4	x28 && !(x30);
x9	F_PU1	87 - 121	!(x31);
x10	S_PU1	1	!(x29 U x31);
x12	S_PU2	0 & 1	!x5;
x13	F_PU3	0	X(x2);
x16	S_PU4	0 & 1	(x5 U x10);
x19	F_PU6	0	F(x8 && x7);
x20	S_PU6	0	(! (x21 U x12));
x21	F_PU7	0 - 51	X(X(x22));
x22	S_PU7	0 & 1	(x19 → (x20 U x19));
x28	S_PU10	0 & 1	X(X(X(x9)));
x29	F_PU11	0	
x30	S_PU11	0	
x31	F_V2	0	

Table 4: Table showing the Case study 2 LTL propositions, affected features of the dataset, their range of values (Inliner) representing the normal states and the generated LTL formula

search conducted by Wang et al. [44]. The study is centred on the utilization of the IEC 61850 standard, which offers an economically efficient software-based simulation platform. This testbed is hosted on Oracle VirtualBox and

comprises five virtual machines (VMs). Among these VMs, one is responsible for emulating a scaled-down primary plant within a distribution substation, employing MATLAB/Simulink. The emulated primary plant encompasses various components, including a 66kV high-voltage line, two transformers, a 22kV low-voltage line, four feeders, and multiple circuit breakers.

The dataset obtained from the simulated test environment encompasses two distinct benign operational behaviours observed in substation operations: normal operation, characterized by the absence of any unusual events, and emergency operation, which occurs during non-malicious events. A total of 31 datasets were generated based on various scenarios, including 15 benign scenarios and 16 attack scenarios. The datasets consist of benign behaviour data, which encompass 7,447 individual samples for normal operation and 12,457 individual samples for emergency operation. Attack datasets include scenarios involving intelligent electronic devices (IEDs), specifically IED1 and IED2, with 8,015 and 9,902 individual samples, respectively.

According to the paper [44], the analysed attacks encompass FDIA and replay attacks. The study involved the creation of eight unique attack scenarios about GOOSE messages generated by IED1, and these scenarios were subsequently replicated for GOOSE messages originating from IED2. Modification of data in the original message, and addition of fake data or fake messages were the two forms of FDIA carried out and four attack scenarios for both the normal operation and the emergency operation respectively were implemented. All benign behaviour datasets and attack datasets from IED1 were used exclusively for training, while the attack datasets from IED2 were reserved solely for testing. Each training and testing dataset was further categorized as follows: two instances of false data injection attacks occurring during normal operations were labelled as 901 and 903; two occurrences of message modification attacks during normal operations were labelled as 902 and 904; two instances of false data injection attacks during emergency operations were labelled as 905 and 907; and two cases of message modification attacks during emergency operations were labelled as 906 and 908.

Data Pre-processing. Our learning LTL formulae process for this case study begins with an initial step: data pre-processing. In this phase, we remove categorical data presented as text. Employing a feature selection approach (using SelectKBest with ANOVA F-statistic), we encapsulate critical information, thereby enhancing the model's capacity to discern patterns within the data. This transformation resulted in a streamlined dataset, reducing its

dimensionality from 36 columns to a more manageable 13 features comprising of network features ("Heartbeat", "APPID", "gocbRef", "goID", "Dif_Sq", "stNum", "sqNum", "num of data", "data_Dec", "data") and physical features ("I_CB-TRSF1", "I_FDR1", "I_FDR4"). After the data pre-processing stage, we proceed to the pattern discovery phase.

Pattern Discovery and Featuring Engineering. Figures 5 to 8 present a visual representation of the statistical distribution of the water distribution system dataset. Following the extraction of significant features through a feature selection mechanism, our focus shifts to discovering patterns in the dataset which we leverage on the labelled column ('Label'). We rely on dataset labelling, assuming that the anomaly detection model proposed by Wang et al. [44] has effectively identified anomalies in the dataset.

The next phase is the application of domain expert analysis, specifically, we extracted the 'positive traces,' and 'negative traces'. The positive traces include n states preceding the malicious events to identify temporary patterns before the occurrence of an attack event. In contrast, the negative traces comprise m states (before and after the occurrence of an attack event) indicative of benign occurrences. The values for n and m are indicated in Table 1.

Consequently, we move forward with the conversion of variables into binary representation using the One-hot encoding. The binary representation of variables sets the groundwork for pattern extraction into a trace file, which serves as the input for the subsequent phase—the learning algorithm for LTL formulas.

LTL Property Learning. Following the extraction and binary representation of positive and negative traces, our focus shifts to the LTL (Linear Temporal Logic) formula learning algorithm, a pivotal phase in our cybersecurity framework. The positive and negative traces, now in a binary format, are seamlessly integrated into the LTL learning algorithm. This algorithm systematically analyses the binary traces to derive LTL formulas that succinctly capture the temporal relationships and patterns associated with each attack scenario. The eight distinct attack scenarios (901, 902, 903, 904, 905, 906, 907, and 908) are individually subjected to the LTL learning process. The dataset characteristics are expressed as propositions (x0, x5, ...) within the LTL formula, and the diverse representations of these features are illustrated in Table 5.

Proposition	$x0$	$x4$	$x5$	$x6$	$x8$	$x9$	$x11$	$x12$	$x13$
Features	Heartbeat	time	Dif_Sq	num_of_data	I_TRSF1-W2	I_CB-TRSF1	I_TRSF1-W1	I_FDR1	I_FDR4

Table 5: The propositions of the LTL formula and the corresponding column names they represent.

Table 6 presents the generated LTL formula for each of the aforementioned attack scenarios. These formulas serve as a representation of the identified temporal patterns.

Class	**901**	**902**	**903**	**904**	**905**	**906**	**907**	**908**
LTL Formula	$F(x0)$	$\neg(F(x4))$	$F(x0)$	$G(x4 \rightarrow x6)$	$F(x13)$	$F(x9)$	$F(x11)$	$F(x12)$
	$F(x5)$	$x6$ U $x12$	$F(x5)$		$\neg(x9)$	$F(x8)$		
		$X(X(x13))$	$G(x9)$					
		$G(\neg(x4))$						

Table 6: Formulated LTL formula from Power Substation Automation Systems Dataset for each attack scenario (901-908).

After generating the LTL formulae, we set the security property according to the formula learned for each testing dataset and perform the runtime monitoring to validate the system program. The property is used to track if there is any execution error that traditional testing or static analysis may not find.

Runtime Monitoring. Using attack datasets of IED2 as the testing, a runtime model is developed. The behaviour of the system at time series is generated as a sequence of states where each state represents the system's state at a specific point in time. In this phase, the testing data set is split into traces based on four (4) states per trace, where each row is a state of the system in a time series. Each trace is fed into the runtime checker.

In runtime monitoring, the compliance of the system via each trace is verified against the set property using the PAT model checker. During the execution of the model checker, we continuously monitor the observed behaviour of the system, checking whether the observed behaviour satisfies the specified LTL formula. This involves evaluating the LTL formula over the

observed behaviour to determine whether it is true or false. The outcome of runtime verification can be described as follows: If the checker returns true, it indicates that the system's behaviour conforms to the specified property, thereby verifying the presence of an anomaly and confirming that a violation has occurred. This implies that the system's behaviour demonstrates that an anomalous event is imminent. Conversely, if the checker returns false, it signifies that no violation has taken place.

5.3.1. Result Analysis

By employing a sequential classification approach based on a sliding window, Wang et al. [44]'s anomaly detection method successfully identified insider attacks on a range of devices, achieving an impressively low false-negative rate of under 1%. Our research goes a step beyond the work conducted by Wang et al. [44]. In contrast, our framework delves into predicting transient attack patterns and formulates a system security property using a Linear Temporal Logic (LTL) formula. This formula is then used to assess the system's compliance in real-time, thus proactively avoiding potential attacks. Table 7 is the case studies' evaluation result.

We measured the prediction time and other metrics. The first prediction occurred 5 states before the anomaly for the class 901 dataset and presented in Table 7 is the prediction time for other datasets. A precision of at least 60% was achieved and this percentage indicates how precise the model is in identifying positive instances. Although class 901 exhibited a low precision rate, higher precision is preferred to minimize false positives, as demonstrated in other cases presented in Table 7. The framework performs better in the other case studies.

Sensitivity values are relatively high, at least 70% across the classes. This suggests that the model is effective at capturing most of the actual positive instances, which is crucial for identifying potential attacks. F1 scores are relatively high, with at least 60% in all classes. This suggests a good balance between precision and recall, indicating that the model is performing well in terms of both false positives and false negatives.

6. Discussion

We demonstrated our work by applying the framework to three case studies in cyber-physical systems and predicting anomaly events within their

datasets. From the training data, we generated system-specific Linear Temporal Logic (LTL) formulas, which were subsequently fed into the runtime monitoring process. The runtime monitoring process involves feeding chunks of testing data as traces (which parameter as indicated in Table 1) into the runtime checker, using the derived LTL formula from the training data for each case study as the specification of properties. This process triggers a "True" or "False" outcome, indicating whether the system conforms to the specified properties or not.

From the outcome of the runtime verification, we evaluated each case study by calculating the prediction time and evaluation metrics such as precision, sensitivity, and the F1 score based on the confusion matrix. The table 7 presents the outcome of the evaluation of the three case studies using our framework to predict the anomaly. The experimental results from all case studies indicate that the model performs well in all cases except for classes 2 and 4 in case study 3.

In Case Study 1 (Section 5.1), out of the total testing data, there were 20 anomaly traces. Based on our definition of precursor events, we would anticipate finding 20 instances or traces of these precursor events. From the runtime verification, 2 traces were false positive (FP) while no false negative trace was recorded. This produced an accuracy score of 0.999 with a precision score of 0.993. This simply means that the framework adequately predicted most instances of anomaly within the dataset. All anomaly events in the testing data of the case study were captured.

In Case Study 2 (Section 5.2), false negatives were observed. Out of all the traces extracted from the testing dataset, 35 attack events were present, so we anticipated finding 35 precursor events. The evaluation uncovered 2 false positive traces and 2 false negative traces. This means that 2 normal cases were incorrectly identified as anomalies, while another 2 instances of anomalies were incorrectly predicted as the negative class.

In Case Study 3 (Section 5.3), across all 8 classes of datasets ranging from 901 to 908, there were recorded rates of 20% false positives and 10% false negatives. The precision, sensitivity, and F1-score for classes 902 and 904 are low due to a high number of false positives and false negatives. These issues can arise due to several factors, including data attack characteristics, noisy data, inappropriate threshold settings, and complex decision boundaries. In our future work, we will focus on addressing these issues by implementing solutions such as data filtering and precise data scaling.

The evaluation results have demonstrated the effectiveness and perfor-

Case Studies	Prediction Time (rows)	Precision	Sensitivity	F1-Score
Case 1: CubeSat	15	1.0	1.0	1.0
Case 2: BATADAL	4	0.91	0.94	0.93
Case 3: Class 901	5	0.908	0.957	0.932
Case 3: Class 902	3	0.604	0.800	0.688
Case 3: Class 903	5	0.883	0.932	0.888
Case 3: Class 904	2	0.656	0.700	0.677
Case 3: Class 905	4	0.888	0.943	0.915
Case 3: Class 906	3	0.961	0.956	0.958
Case 3: Class 907	5	0.872	0.928	0.899
Case 3: Class 908	3	0.913	0.930	0.921

Table 7: Evaluation Result of the three case studies. Case 1 is the CubeSat dataset and the anomaly is a false data injection attack, Case 2 is an anomaly in the Water distribution system (BATADAL) [41]. Case 3 is anomalies in Substation Automation Systems [44] where class 901, 903, 905, and 907 are different types of replay attacks and class 902, 904, 906 and 908 are different types of false data injection attacks.

mance of our framework; however, several limitations impact its prediction accuracy. One key limitation is the framework's dependence on data clustering/classification, which plays a crucial role in identifying patterns and making predictions. The accuracy of the predictions is highly influenced by the quality and precision of the clustering process, which may not always be optimal for all types of data or scenarios.

Furthermore, the determination of which clusters correspond to normal, precursor and abnormal behaviours is subjective and depends on the expertise of domain specialists or the setting of a threshold. This introduces another layer of limitation, as the choice of these labels can vary across different domains or contexts.

6.1. Comparison with other models

The evaluation results indicate that the framework can predict the anomaly, though we face evaluation challenges in comparing our result outcomes with other related works. To assess the performance of the proposed framework, the latter is compared with other models [16, 18, 19, 25, 40].
Existing anomaly prediction models depend on machine learning algorithms such as the Logistic Regression algorithm [19], Isolation Forest (IF) algorithm [16], Decision Trees [40] while for Interpretable Anomaly Prediction (IAP) [25], probabilistic anomaly score was determined using regularized logistic regression. However, there are some deficiencies both in application and function. In terms of function, all these models lack the ability to generate alerts or provide timely responses on predicting the anomalies in real-time while our framework which combines LTL with machine learning in a hybrid approach addresses the problem making it an advanced state-of-the-art model. In terms of application, the input and output data used in the models of El-Hadad et al. [16], Gu and Wang [18], Hadj-Kacem et al. [19], and Tan et al. [40] differ from the data employed in our model, making a direct comparison with our work challenging.

Furthermore, two of the datasets (BATADAL [41] and Power Substation Automation Systems [44]) deployed in our study were originally used for anomaly detection models and not for prediction. We analysed the datasets for anomaly prediction with the assumption that there will exist a predictive pattern leading to the anomaly events. Our framework extends beyond detecting anomaly events; it is designed to predict the events before the actual occurrence, thereby serving as a preventive measure against the occurrence of attacks. We introduced an LTL-based runtime verification technique to predict anomaly symptoms by applying LTL as system-defined properties which are human-readable rules, making it easy to interpret flagged anomalies based on expected temporal patterns. Violations of specific LTL properties directly explain anomalies, allowing precise identification of cause and context. Based on our approach, the LTL properties can be tailored to capture the unique operational behaviours, and failure modes of a specific system and designed with a special alert state, as illustrated in Figure 2 to ensure a prompt response can be made before impact.

6.2. Experimental Evaluation Challenges

We recognise some challenges in evaluating the outcome of our experimental results with related research. First, our datasets were generated within a

simulated environment rather than being collected from an actual real-time Cyber-Physical System (CPS). The absence of experiments conducted with real CPS hinders our ability to conclude on the performance of our framework. At this stage, additional experiments are imperative to assert any generalized claims of the framework to other CPS scenarios.

Second, in the experimental evaluation, two out of the three datasets, BATADAL [41] and Power Substation Automation System [44] were sourced online, while the CubaSat dataset was internally generated. Notably, these datasets were primarily used for anomaly detection assessment rather than prediction, and the CubaSat dataset has not been publicly released. Consequently, as per our current knowledge, none of these datasets has been previously employed for anomaly prediction purposes. This lack of comparable prior work limits the available benchmarks for evaluating our framework.

Third, we applied various algorithms for data pre-processing, clustering, and classification. Our anomaly prediction hinges on the outcomes produced by these algorithms. The selection of the algorithm depends on the specific characteristics and nature of the data.

7. Conclusion

Safeguarding users and valuable assets from cyber-attacks is one of the most important challenges within cybersecurity. Particularly, the tampering of engineering data in cyber-physical systems and IoT systems can severely harm the system's integrity and its users' well-being.

To address these concerns, we have introduced an LTL based runtime verification framework that leverages runtime verification, effectively using the strengths of data analytics and the acquisition of Linear Temporal Logic (LTL) formulas. This paper delves into the methodology for extracting LTL formulas from historical data. Our approach involves the processing of historical data, employing clustering techniques to derive sequences that encapsulate the evolving pattern. These data sequences are subsequently fed into an LTL learning algorithm, resulting in the creation of LTL formulas that encapsulate the detected patterns. An important feature of our framework is its unique capability to proactively predict anomalies, encompassing both faults and potential attacks, well in advance of their actual manifestation.

This entire procedure seamlessly integrates into our runtime framework, resulting in a comprehensive and holistic approach. Our approach effectiveness is validated through extensive evaluations employing three case studies

datasets from cyber-physical systems, where we consistently achieve an impressive accuracy rate of at least 90% across the three cases in pre-emptively forecasting anomalies within time series data.

References

[1] C. C. Aggarwal and C. C. Aggarwal. *An introduction to outlier analysis*. Springer, 2017.

[2] M. Aghashahi, R. Sundararajan, M. Pourahmadi, and M. K. Banks. Water distribution systems analysis symposium–battle of the attack detection algorithms (batadal). In *World Environmental and Water Resources Congress 2017*, pages 101–108, 2017.

[3] S. Ahmed, Y. Lee, S.-H. Hyun, and I. Koo. Unsupervised machine learning-based detection of covert data integrity assault in smart grid networks utilizing isolation forest. *IEEE Transactions on Information Forensics and Security*, 14(10):2765–2777, 2019.

[4] B. A. Alkhaleel. Machine learning applications in the resilience of interdependent critical infrastructure systems—a systematic literature review. *International Journal of Critical Infrastructure Protection*, 44:100646, 2024. ISSN 1874-5482. doi: https://doi.org/10.1016/j.ijcip.2023.100646. URL `https://www.sciencedirect.com/science/article/pii/S1874548223000598`.

[5] A. Alqudhaibi, M. Albarrak, A. Aloseel, S. Jagtap, and K. Salonitis. Predicting cybersecurity threats in critical infrastructure for industry 4.0: a proactive approach based on attacker motivations. *Sensors*, 23 (9):4539, 2023.

[6] R. Baheti and H. Gill. Cyber-physical systems. *The impact of control technology*, 12(1):161–166, 2011.

[7] E. Bartocci, Y. Falcone, A. Francalanza, and G. Reger. Introduction to runtime verification. *Lectures on Runtime Verification: Introductory and Advanced Topics*, pages 1–33, 2018.

[8] M. Bashendy, A. Tantawy, and A. Erradi. Intrusion response systems for cyber-physical systems: A comprehensive survey. *Computers & Security*, 124:102984, 2023. ISSN 0167-4048. doi: https://doi.org/10.1016/

j.cose.2022.102984. URL https://www.sciencedirect.com/science/article/pii/S0167404822003765.

[9] T. Berghout, M. Benbouzid, and S. Muyeen. Machine learning for cybersecurity in smart grids: A comprehensive review-based study on methods, solutions, and prospects. *International Journal of Critical Infrastructure Protection*, 38:100547, 2022. ISSN 1874-5482. doi: https://doi.org/10.1016/j.ijcip.2022.100547. URL https://www.sciencedirect.com/science/article/pii/S1874548222000348.

[10] A. Bernstein and P. K. Harter Jr. Proving real-time properties of programs with temporal logic. *ACM SIGOPS Operating Systems Review*, 15(5):1–11, 1981.

[11] J. Chen and X. Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[12] K. Choi, J. Yi, C. Park, and S. Yoon. Deep learning for anomaly detection in time-series data: Review, analysis, and guidelines. *IEEE Access*, 9:120043–120065, 2021. doi: 10.1109/ACCESS.2021.3107975.

[13] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. Formalization and validation of safety-critical requirements. *arXiv preprint arXiv:1003.1741*, 2010.

[14] B. Dou, Z. Zhu, E. Merkurjev, L. Ke, L. Chen, J. Jiang, Y. Zhu, J. Liu, B. Zhang, and G.-W. Wei. Machine learning methods for small data challenges in molecular science. *Chemical Reviews*, 123(13):8736–8780, 2023.

[15] X. Du, A. Tiu, K. Cheng, and Y. Liu. Trace-length independent runtime monitoring of quantitative policies. *IEEE Transactions on Dependable and Secure Computing*, 18(3):1489–1510, 2019.

[16] R. El-Hadad, Y. F. Tan, and W. N. Tan. Anomaly prediction in electricity consumption using a combination of machine learning techniques. *International Journal of Technology*, 13(6):1317–1325, 2022.

[17] Y. Falcone, K. Havelund, and G. Reger. A tutorial on runtime verification. *Engineering dependable software systems*, pages 141–175, 2013.

[18] X. Gu and H. Wang. Online anomaly prediction for robust cluster systems. In *2009 IEEE 25th International Conference on Data Engineering*, pages 1000–1011. IEEE, 2009.

[19] I. Hadj-Kacem, S. B. Jemaa, S. Allio, and Y. B. Slimen. Anomaly prediction in mobile networks : A data driven approach for machine learning algorithm selection. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7, 2020. doi: 10.1109/NOMS47738.2020.9110429.

[20] K. Havelund and D. Peled. Runtime verification: from propositional to first-order temporal logic. In *Runtime Verification: 18th International Conference, RV 2018, Limassol, Cyprus, November 10–13, 2018, Proceedings 18*, pages 90–112. Springer, 2018.

[21] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[22] Z. Hou, Q. Li, E. Foo, J. Song, and P. Souza. A digital twin runtime verification framework for protecting satellites systems from cyber attacks. In *2022 26th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 117–122. IEEE, 2022.

[23] J. Hu, B. Lennox, and F. Arvin. Robust formation control for networked robotic systems using negative imaginary dynamics. *Automatica*, 140: 110235, 2022.

[24] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor. Unsupervised learning. In *An Introduction to Statistical Learning: with Applications in Python*, pages 503–556. Springer, 2023.

[25] R. Langone, A. Cuzzocrea, and N. Skantzos. Interpretable anomaly prediction: Predicting anomalous behavior in industry 4.0 settings via regularized logistic regression tools. *Data & Knowledge Engineering*, 130:101850, 2020. ISSN 0169-023X. doi: https://doi.org/10.1016/j.datak.2020.101850. URL `https://www.sciencedirect.com/science/article/pii/S0169023X1830644X`.

[26] A. Legay, B. Delahaye, and S. Bensalem. Runtime verification. *Lecture Notes in Computer Science*, 6418:122–135, 2010.

[27] T. Li, M. L. Comer, E. J. Delp, S. R. Desai, J. L. Mathieson, R. H. Foster, and M. W. Chan. Anomaly scoring for prediction-based anomaly detection in time series. In *2020 IEEE Aerospace Conference*, pages 1–7. IEEE, 2020.

[28] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, et al. Long short term memory networks for anomaly detection in time series. In *Esann*, volume 2015, page 89, 2015.

[29] D. Neider and I. Gavran. Learning linear temporal properties. In *2018 Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–10. IEEE, 2018.

[30] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee, 1977.

[31] R. Raha, R. Roy, N. Fijalkow, and D. Neider. Scalable anytime algorithms for learning fragments of linear temporal logic. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 263–280. Springer, 2022.

[32] V. S. Rajkumar, A. Ştefanov, A. Presekal, P. Palensky, and J. L. R. Torres. Cyber attacks on power grids: Causes and propagation of cascading failures. *IEEE Access*, 2023.

[33] H. Riggs, S. Tufail, I. Parvez, M. Tariq, M. A. Khan, A. Amir, K. V. Vuda, and A. I. Sarwat. Impact, vulnerabilities, and mitigation strategies for cyber-secure critical infrastructure. *Sensors*, 23(8):4060, 2023.

[34] C. Schaffer. Selecting a classification method by cross-validation. *Machine learning*, 13:135–143, 1993.

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

[36] K. R. Shahapure and C. Nicholas. Cluster quality analysis using silhouette score. In *2020 IEEE 7th international conference on data science and advanced analytics (DSAA)*, pages 747–748. IEEE, 2020.

[37] Z. Shang, Y. Zhang, X. Zhang, Y. Zhao, Z. Cao, and X. Wang. Time series anomaly detection for kpis based on correlation analysis and hmm. *Applied Sciences*, 11(23), 2021. ISSN 2076-3417. doi: 10.3390/app112311353. URL `https://www.mdpi.com/2076-3417/11/23/11353`.

[38] J. Sun, Y. Liu, and J. S. Dong. Model checking csp revisited: Introducing a process analysis toolkit. In *International symposium on leveraging applications of formal methods, verification and validation*, pages 307–322. Springer, 2008.

[39] J. Sun, Y. Liu, J. S. Dong, and J. Pang. Pat: Towards flexible verification under fairness. In *International conference on computer aided verification*, pages 709–714. Springer, 2009.

[40] Y. Tan, X. Gu, and H. Wang. Adaptive system anomaly prediction for large-scale hosting infrastructures. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 173–182, 2010.

[41] R. Taormina, S. Galelli, N. O. Tippenhauer, E. Salomons, A. Ostfeld, D. G. Eliades, M. Aghashahi, R. Sundararajan, M. Pourahmadi, M. K. Banks, et al. Battle of the attack detection algorithms: Disclosing cyber attacks on water distribution networks. *Journal of Water Resources Planning and Management*, 144(8):04018048, 2018.

[42] E. Umargono, J. E. Suseno, and S. V. Gunawan. K-means clustering optimization using the elbow method and early centroid determination based on mean and median formula. In *The 2nd International Seminar on Science and Technology (ISSTEC 2019)*, pages 121–129. Atlantis Press, 2020.

[43] M. A. Umer, C. M. Ahmed, M. T. Jilani, and A. P. Mathur. Attack rules: an adversarial approach to generate attacks for industrial control systems using machine learning. In *Proceedings of the 2th Workshop on CPS&IoT Security and Privacy*, pages 35–40, 2021.

[44] X. Wang, C. Fidge, G. Nourbakhsh, E. Foo, Z. Jadidi, and C. Li. Anomaly detection for insider attacks from untrusted intelligent electronic devices in substation automation systems. *IEEE Access*, 10:6629–6649, 2022. doi: 10.1109/ACCESS.2022.3142022.

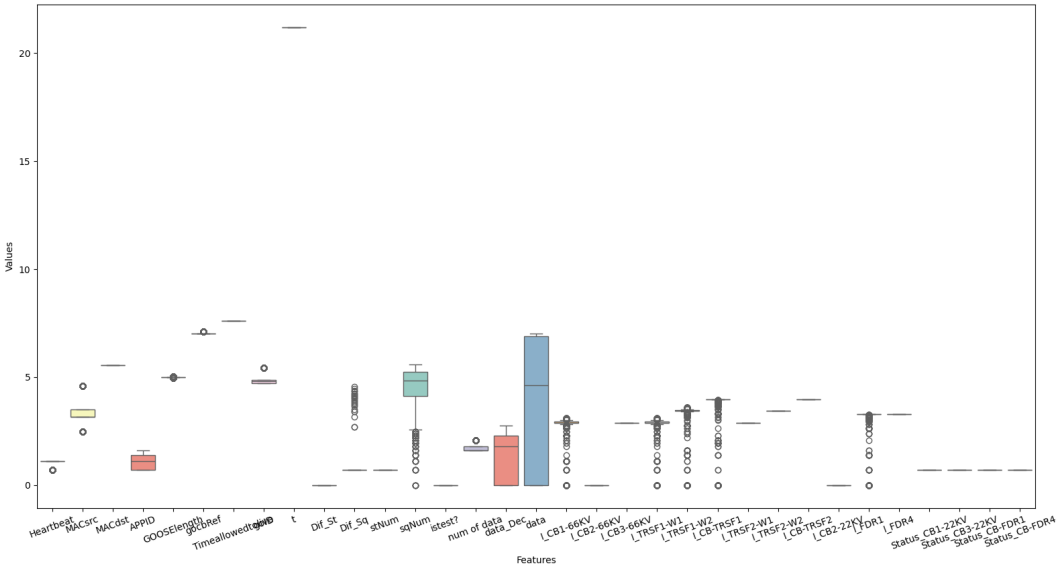
[45] L. Yang. Pat: Process analysis toolkit, 2009. URL `https://pat.comp.nus.edu.sg/`. Accessed: 17 October, 2024.

[46] X.-X. Yin, Y. Miao, and Y. Zhang. Time series based data explorer and stream analysis for anomaly prediction. *Wireless Communications and Mobile Computing*, 2022(1):5885904, 2022.

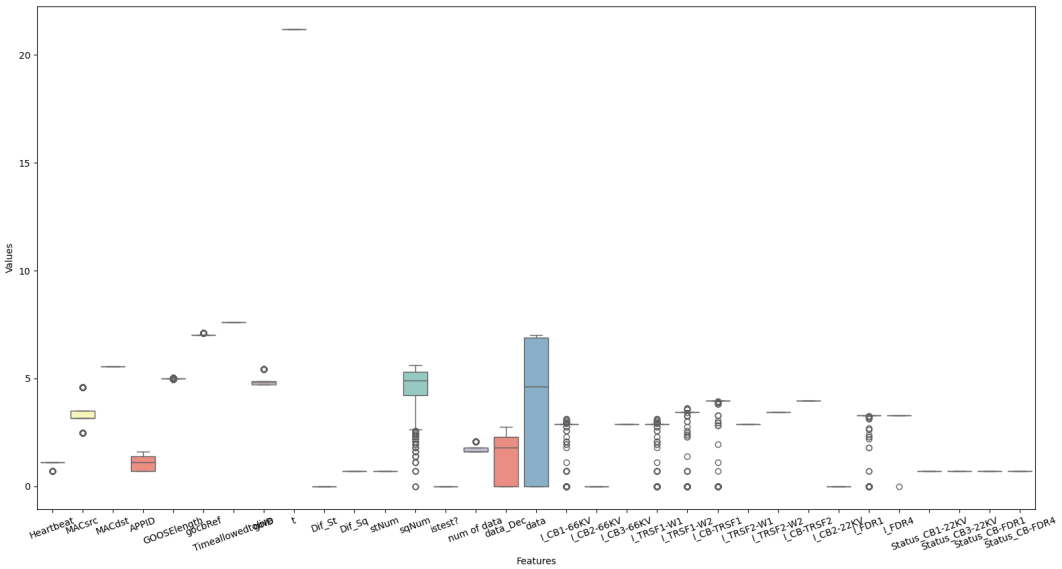
[47] H. Zhang, B. Liu, and H. Wu. Smart grid cyber-physical attack and defense: A review. *IEEE Access*, 9:29641–29659, 2021.

[48] X. Zheng, C. Julien, M. Kim, and S. Khurshid. Perceptions on the state of the art in verification and validation in cyber-physical systems. *IEEE Systems Journal*, 11(4):2614–2627, 2015.

[49] X. Zheng, C. Julien, R. Podorozhny, and F. Cassez. Braceassertion: Runtime verification of cyber-physical systems. In *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 298–306. IEEE, 2015.

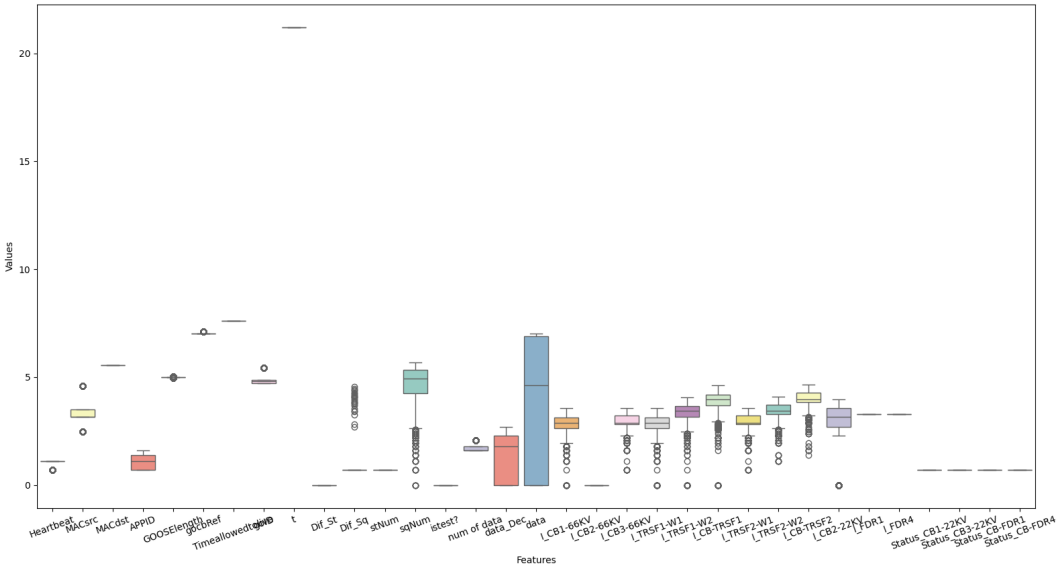


(a) 901

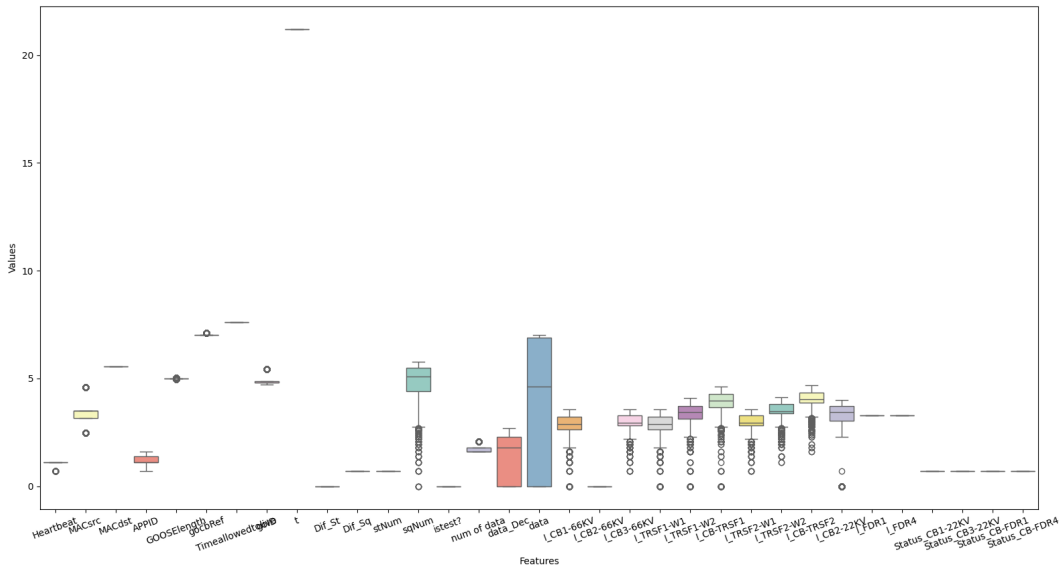


(b) 902

Figure 5: The graph indicates the statistical distribution of data points for the Power Substation Automation Systems dataset for class 901 and 902.

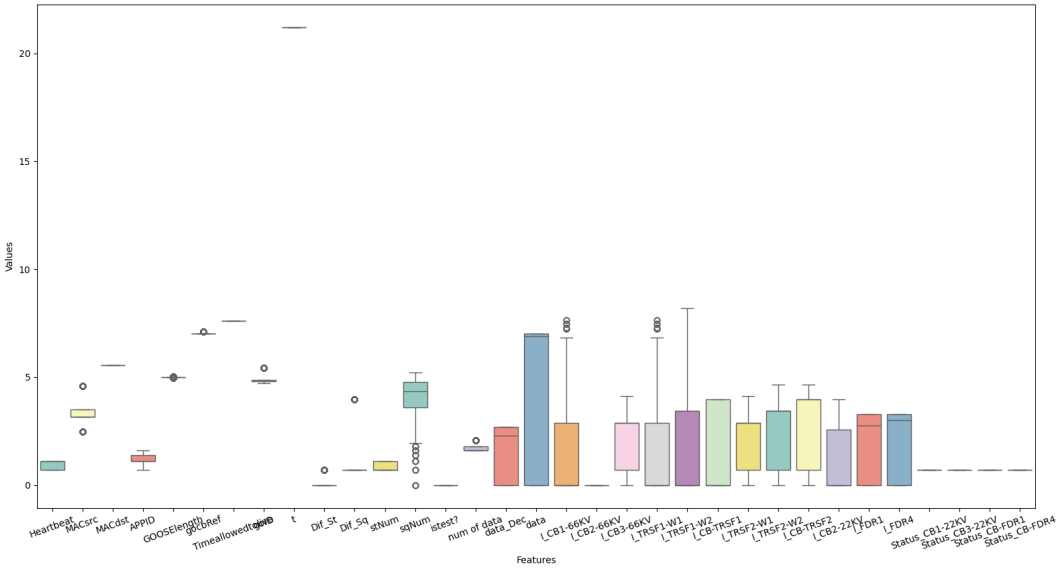


(a) 903

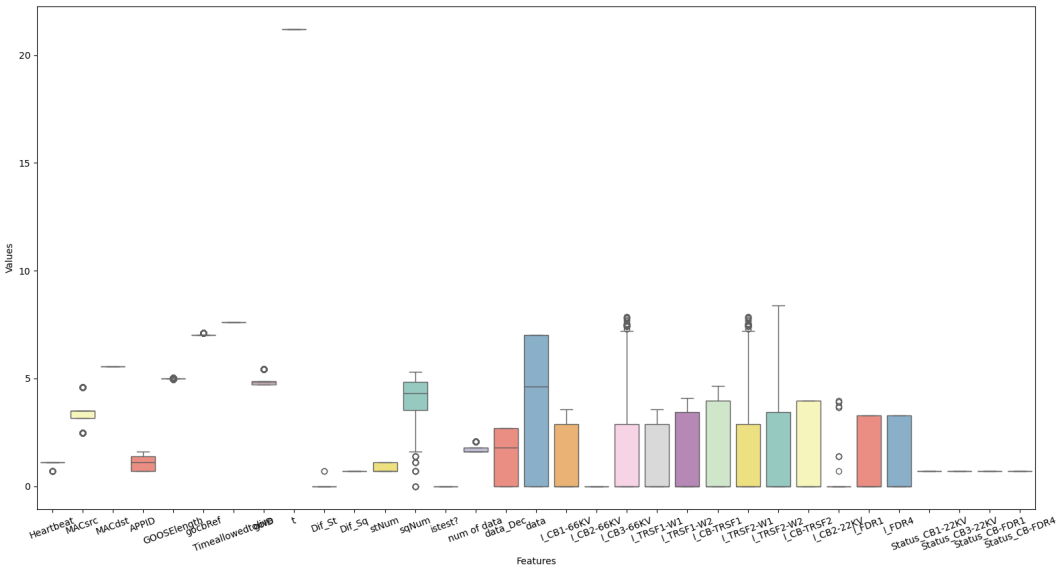


(b) 904

Figure 6: The graph indicates the statistical distribution of data points for the Power Substation Automation Systems dataset for class 903 and 904.

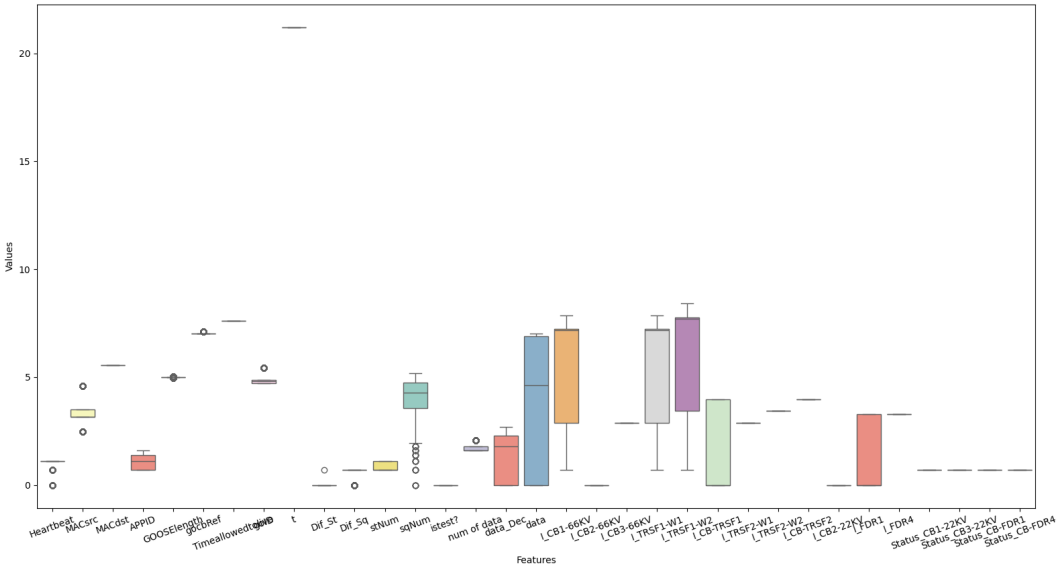


(a) 905

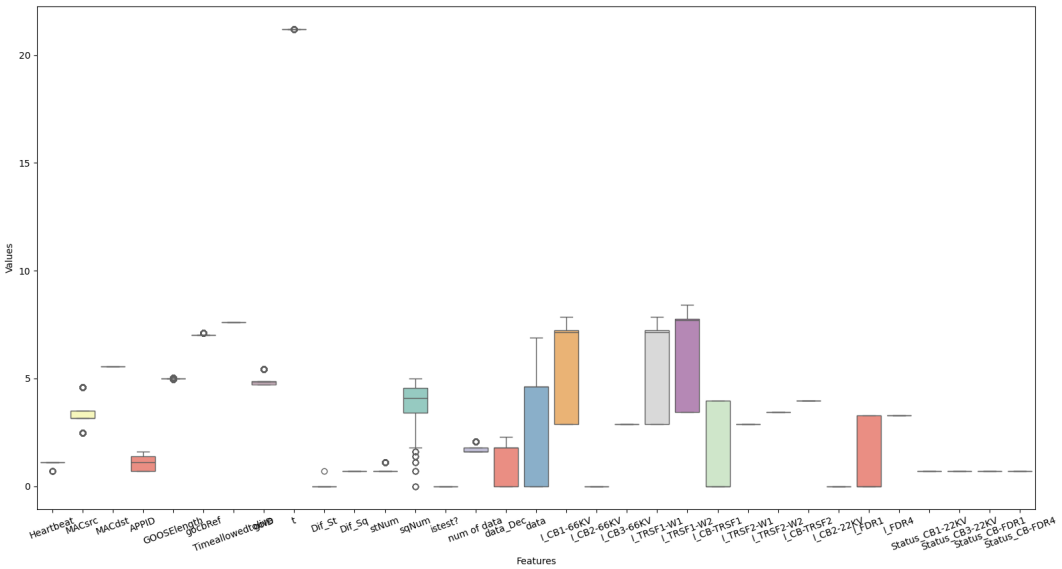


(b) 906

Figure 7: The graph indicates the statistical distribution of data points for the Power Substation Automation Systems dataset for class 905 and 906.



(a) 907



(b) 908

Figure 8: The graph indicates the statistical distribution of data points for the Power Substation Automation Systems dataset for class 907 and 908.