UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Robust Anomaly Detection in Critical Infrastructure

Maged Abdelaty

Advisor

Dr. Domenico Siracusa

Fondazione Bruno Kessler

Co-Advisor

Dr. Roberto Doriguzzi-Corin

Fondazione Bruno Kessler

A thesis submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy in Information and Communication Technology
September 2022

Abstract

Critical Infrastructures (CIs) such as water treatment plants, power grids and telecommunication networks are critical to the daily activities and well-being of our society. Disruption of such CIs would have catastrophic consequences for public safety and the national economy. Hence, these infrastructures have become major targets in the upsurge of cyberattacks. Defending against such attacks often depends on an arsenal of cyber-defence tools, including Machine Learning (ML)-based Anomaly Detection Systems (ADSs). These detection systems use ML models to learn the profile of the normal behaviour of a CI and classify deviations that go well beyond the normality profile as anomalies. However, ML methods are vulnerable to both adversarial and non-adversarial input perturbations. Adversarial perturbations are imperceptible noises added to the input data by an attacker to evade the classification mechanism. Non-adversarial perturbations can be a normal behaviour evolution as a result of changes in usage patterns or other characteristics and noisy data from normally degrading devices, generating a high rate of false positives.

We first study the problem of ML-based ADSs being vulnerable to non-adversarial perturbations, which causes a high rate of false alarms. To address this problem, we propose an ADS called DAICS, based on a wide and deep learning model that is both adaptive to evolving normality and robust to noisy data normally emerging from the system. DAICS adapts the pre-trained model to new normality with a small number of data samples and a few gradient updates based on feedback from the operator on false alarms. The DAICS was evaluated on two datasets collected from real-world Industrial Control System (ICS) testbeds. The results show that the adaptation process is fast and that DAICS has an improved robustness compared to state-of-the-art approaches.

We further investigated the problem of false-positive alarms in the

ADSs. To address this problem, an extension of DAICS, called the SiFA framework, is proposed. The SiFA collects a buffer of historical false alarms and suppresses every new alarm that is similar to these false alarms. The proposed framework is evaluated using a dataset collected from a real-world ICS testbed. The evaluation results show that the SiFA can decrease the false alarm rate of DAICS by more than 80%.

We also investigate the problem of ML-based network ADSs that are vulnerable to adversarial perturbations. In the case of network ADSs, attackers may use their knowledge of anomaly detection logic to generate malicious traffic that remains undetected. One way to solve this issue is to adopt adversarial training in which the training set is augmented with adversarially perturbed samples. This thesis presents an adversarial training approach called GADoT that leverages a Generative Adversarial Network (GAN) to generate adversarial samples for training. GADoT is validated in the scenario of an ADS detecting Distributed Denial of Service (DDoS) attacks, which have been witnessing an increase in volume and complexity. For a practical evaluation, the DDoS network traffic was perturbed to generate two datasets while fully preserving the semantics of the attack. The results show that adversaries can exploit their domain expertise to craft adversarial attacks without requiring knowledge of the underlying detection model. We then demonstrate that adversarial training using GADoT renders ML models more robust to adversarial perturbations.

However, the evaluation of adversarial robustness is often susceptible to errors, leading to robustness overestimation. We investigate the problem of robustness overestimation in network ADSs and propose an adversarial attack called UPAS to evaluate the robustness of such ADSs. The UPAS attack perturbs the inter-arrival time between packets by injecting a random time delay before packets from the attacker. The attack is validated by perturbing malicious network traffic in a multi-attack dataset and used

to evaluate the robustness of two robust ADSs, which are based on a denoising autoencoder and an adversarially trained ML model. The results demonstrate that the robustness of both ADSs is overestimated and that a standardised evaluation of robustness is needed.

Keywords

[Anomaly Detection; Deep Learning; Robustness; Industrial Control System; Adversarial Training; DDoS Attack]

Contents

List of Tables

List of Figures

Acronyms

ADS ..... Anomaly Detection System

AML ..... Adversarial Machine Learning

BFP ..... Benign Feature Perturbation

CI ..... Critical Infrastructure

CNN ..... Convolutional Neural Network

DDoS ..... Distributed Denial of Service

FGSM ..... Fast Gradient Sign Method

FNR ..... False Negative Rate

FPR ..... False Positive Rate

GAN ..... Generative Adversarial Network

ICS ..... Industrial Control System

LOIC ..... Low Orbit Ion Cannon

MLP ..... Multilayer Perceptron

ML ..... Machine Learning

MTU ..... Maximum Transmission Unit

NIDS ..... Network Intrusion Detection System

PGD ..... Projected Gradient Descent

PLC ..... Programmable Logic Controller

PRNG ..... Pseudo Random Number Generator

RePO+ .... Reconstruction from Partial Observation plus

SCADA ... Supervisory Control and Data Acquisition

SWaT ..... Secure Water Treatment

TPR ..... True Positive Rate

TTNN ..... Threshold Tuning Neural Network

WADI ..... Water Distribution

WDNN .... Wide and Deep Neural Network

WGAN-GP Wasserstein GAN with gradient penalty

WGAN .... Wasserstein GAN

To my family.

Acknowledgements

I would like to thank my advisor, Domenico Siracusa, for his unconditional support during my Ph.D. It has been a pleasure working with you. I would like to thank my co-advisor Roberto Doriguzzi-Corin for his endless support and mentoring. I would also like to thank Sandra Scott-Hayward for encouraging discussions and comments that led to the development of part of this work. I would like to thank my colleagues, past and present, from the Robust and Secure Distributed Computing (RiSING) research unit for their support and friendship.

Chapter 1

Introduction

1.1 Motivation

Critical Infrastructures (CIs) such as water treatment plants, power grids and telecommunication networks are vital to ensure the security and well-being of citizens. For instance, water treatment plants are CIs that use networked control systems to automate operations, such as water flow rates and chemical additions, enabling the treatment of large amounts of water. The disruption of such CIs can threaten public health and result in economic losses. Hence, these infrastructures have become major targets of cyberattacks recently.

Examples of celebrated attacks on CIs include cyberattacks against the Ukrainian power grid and the Mirai DDoS attack. In an attack against the Ukrainian power grid, the attackers exploited the BlackEnergy (BE) malware to compromise the corporate network assets of three regional power distribution companies [1]. The attackers then gained access to Supervisory Control and Data Acquisition (SCADA) workstations and executed malicious code to alter the firmware of specific control devices and initiate unscheduled disconnections from servers. The attack affected thousands of users and left them without electricity.

The Mirai DDoS attack started in September 2016 when a massive wave

of DDoS attacks targeted, among others, the domain name system provider Dyn using malware called Mirai [2]. According to Antonakakis et al. [3], Mirai infected more than 200,000 Linux-based Internet of Things (IoT) devices that used either weak or default passwords. The infected devices were employed to build several botnets to execute unprecedented DDoS attacks. The attacks reduced the availability of many online services that rely on the Dyn service provider, particularly on the east coast of the US. The affected websites included Twitter, PayPal, and, to some extent, Amazon [3].

Protecting CIs against cyberattacks requires sophisticated tools for network security monitoring including ADSs. ADSs raise alerts when the controlled process deviates from its normal behaviour to an unexpected state. These detection systems define a model of normal behaviour by employing ML and Deep Learning (DL) models. In an online system, they monitor the status of the system to detect any deviation from the predefined normal behaviour, i.e., an unusual or unexpected behaviour, caused by an attack or a malfunctioning device. The deviation is quantified and compared to anomaly rules in the form of thresholds or upper and lower limits [4]. ADSs have the advantage of detecting threats that do not have known signatures, such as zero-day vulnerabilities and novel attack techniques. It is worth noting that most CI attacks exploit zero-day vulnerabilities. Furthermore, ADSs do not interfere with the monitored system, making them ideal for legacy systems that can crash easily under active security measures [5, 6].

However, ML methods are vulnerable to both adversarial and non-adversarial input perturbations [7]. Adversarial perturbations are imperceptible noises added by an attacker to alter the decision of a classification mechanism. It was demonstrated in [8–10] that an attacker can perturb malicious network traffic to remain undetected with limited or no knowledge of

the underlying ML model. In contrast, non-adversarial perturbations can be natural noise that emerges from ageing devices, perturbing the input of the classification mechanism, triggering false positives and thus decreasing the accuracy. For instance, [11–13] demonstrated that state-of-the-art classifiers are vulnerable to additive Gaussian noise. Based on the aforementioned examples, it can be said that the ML techniques are not robust to input perturbations regardless of the perturbations source. Therefore, there is a research gap to fill in the area of resilient ML models.

Given the importance of ML-based ADSs for CI security, it is imperative to work through their limitations. In this thesis, we propose ML-based ADSs that are robust to both adversarial and non-adversarial input perturbations.

1.2 Challenges

As mentioned earlier, ML methods are vulnerable to non-adversarial input perturbations, i.e., perturbations from natural sources, such as normal behaviour evolution and additive Gaussian noise [11–13]. A normal behaviour evolution can occur due to changes in usage patterns or product re-engineering, causing the input data to deviate from the known normal behaviour. The tangible result of such deviations is an increasing number of false positives as novel behaviours are considered abnormal. For instance, in a water treatment plant, increasing the capacity of a water tank would change the behaviour of the sensors attached to the tank, causing false positives until the ADS adapts to the novel behaviour. Another important source of non-adversarial perturbations is the additive Gaussian noise that emerges from normally degrading devices. This adds noise to data readings and thus generates additional false positives. This high rate of false positive alarms hampers the work of ADSs. Therefore, an important challenge is to propose an ADS that is robust to non-adversarial input perturbations to

reduce the number of false alarms.

ML methods can also be vulnerable to adversarial perturbations in which the adversary adds imperceptible perturbations to the input to alter the decision of the ML model. We consider an evasion scenario in which the attacker perturbs the malicious input of a Network Intrusion Detection System (NIDS) to force the underlying ML model to label such an input (i.e., network traffic) as benign. The application domain of network intrusion detection constrains the process of generating adversarial traffic as such traffic must maintain the malicious behaviour and remain valid according to the network protocols. The potential of ML-based NIDSs to misclassify adversarial traffic and hence reduce NIDS accuracy has been demonstrated in [8–10]. For example, Sadeghzadeh et al. [9] presented an approach for generating adversarial traffic by injecting perturbations at the beginning or end of the payload of malicious packets, or by sending dummy packets at the end of malicious network flows. Aiken et al. [8] proposed an adversarial attack in which features such as payload size and packet rate are perturbed. The results suggest that such perturbations can be implemented in live traffic without compromising the attack function. Therefore, it is challenging to propose a NIDS that is robust against adversarial input perturbations.

1.3 Contributions

The contributions of this thesis are as follows:

- The problem of robustness of ML models to non-adversarial input perturbations is studied in Chapter 3. We propose a novel framework called Deep learning Anomaly detection in Industrial Control Systems (DAICS), based on a wide and deep neural network that learns the normal behaviour of the system [14, 15]. The neural network has a

modular architecture that is suitable for large systems. In addition, DAICS includes an algorithm called *few-time-steps* that adapts the model of normality to the latest change in normal behaviour based on feedback from the operator (i.e., false positive alarms). Therefore, the ADS is robust to perturbations from a normal behaviour evolution. Deviations from normality are classified as anomalies based on the threshold applied to the residuals between the model predictions and observations collected from the real system. An automatic threshold tuning technique is presented to dynamically tune the detection threshold based on the prediction error observed in the system. The sensitivity of the proposed approach to additive Gaussian noise, emerging in the system from the degradation of devices in the system and interference, is also measured. The evaluation is performed using two datasets, Secure Water Treatment (SWaT) and Water Distribution (WADI), collected from real-world ICS testbeds. The sensitivity of the hyperparameters of the proposed ADS is analysed to offer insights into the tuning of these hyperparameters to further reduce the number of false alarms and, hence, the burden on the operator.

- The problem of false positive alarms is further studied in Chapter 4. We propose a framework called Similarity based technique for reduction of False Alarms (SiFA), which is an extension of DAICS, for the reduction of false positive alarms in ADSs that are monitoring ICSs. The SiFA collects a buffer of historical false alarms and suppresses every new alarm similar to these false alarms. In other words, SiFA maintains a buffer of false alarms, and when the DAICS anomaly detection logic raises a new alarm, the similarity between this alarm and every instance in the buffer is computed using the squared Euclidean distance. The computed minimum distance is compared with a similarity threshold. If the computed minimum distance is above the threshold, the new

alarm is considered dissimilar to the historical false alarms and is hence reported to the operator. If the computed minimum distance is below this threshold, the alarm is considered similar to at least one historical false alarm and is hence suppressed. The SiFA is evaluated using a dataset collected from an ICS testbed. The results show that the SiFA can decrease the false alarm rate of DAICS by more than 80%.

- The problem of adversarial robustness of ML models is studied in Chapter 5. We propose a novel approach called GAN-based Adversarial training for robust DDoS attack deTection (GADoT) to increase the robustness of ML models against adversarial input perturbations [16]. GADoT exploits a GAN trained on benign samples to generate fake-benign samples to perturb the malicious samples. These perturbed samples are used for adversarial training, which is conducted using an augmented training dataset comprising benign samples, malicious samples and perturbed malicious samples. A practical evaluation is performed based on network traffic traces that capture adversarially perturbed DDoS attacks while preserving the semantics of the attack. Experiments show that attackers can use their domain expertise to generate perturbed malicious traffic that remains undetected without requiring knowledge of the victim ML model. Single and multiple (combined) feature perturbations in network traces are further analysed to offer insight into the robustness of the ML model before and after training with the GADoT. The results show that models trained with GADoT can detect the malicious network flows regardless of perturbation, as the undetected malicious flows decrease to approximately 1% after adversarial training.

- The problem of adversarial robustness overestimation of ML models is studied in Chapter 6. We propose an adversarial attack called

Universal Perturbation Attack against robust NIDSs (UPAS) to evaluate the robustness of ML models that are resistant to adversarial input perturbations. The UPAS attack perturbs the inter-arrival time between packets through injecting a random time delay before the packets sent by attacker. The attack activity is injected in the malicious traffic of the CICIDS2017 dataset to evaluate the robustness of two robust NIDSs, which are based on a denoising autoencoder and an adversarially trained ML model. The results show that the robustness of both NIDSs is overestimated. Subsequently, a simple adversarial training step (without using a GAN) is proposed to improve the robustness against the UPAS attack.

1.4 Structure of the thesis

The technical contributions of this thesis are presented in Chapters 3 to 6. The remainder of this thesis is organised as shown in Figure 1.1.

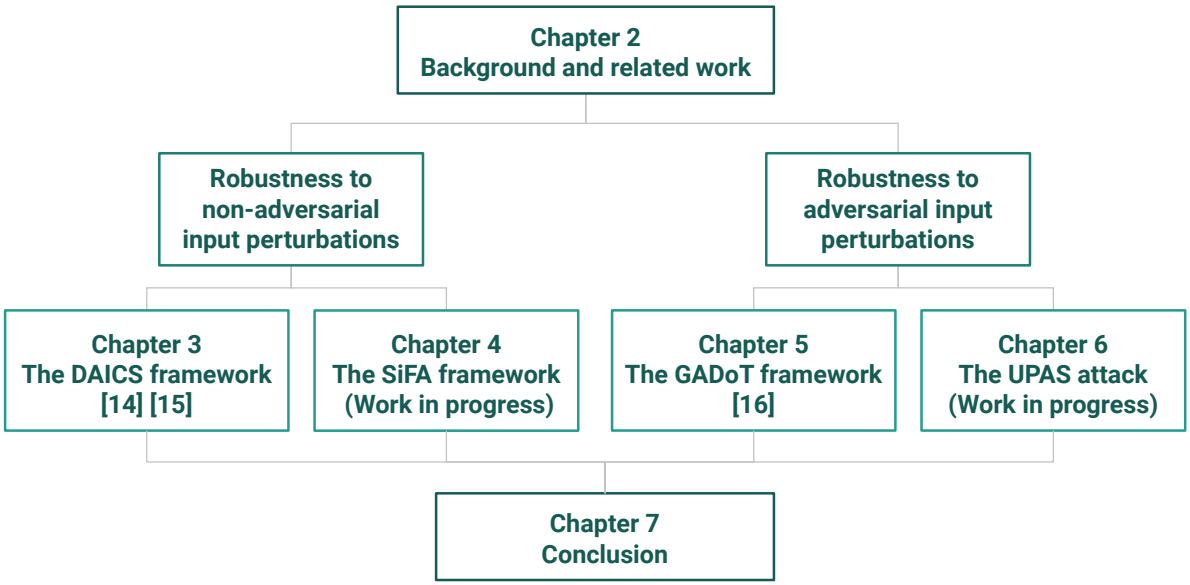

Figure 1.1: Illustration of the thesis structure.

Chapter 2

Background and Related Work

This chapter provides an overview of the works related to the contributions of the thesis. In Section 2.1, we give a brief introduction to the problem of robustness to non-adversarial input perturbations in the context of industrial control systems. In Section 2.1.1, we start with providing a background on the ICSs and examples of recent attacks targeting them, as a motivation. Then, two ICS testbeds and the datasets collected from each of them are described and examples of non-adversarial input perturbations in both datasets are presented. In Section 2.1.2, the ML and DL methods employed to detect anomalies in ICSs are reviewed; the limitations are identified and discussed. Finally, approaches most relevant to this thesis are compared to highlight our contributions in Section 2.1.3.

In Section 2.2, we briefly introduce the problem of vulnerability to adversarial perturbations in the case of ML-based NIDSs detecting DDoS attacks. In Section 2.2.1, we start with presenting a background to DDoS attack types and detection systems. In Section 2.2.2, the adversarial attacks aiming to evade ML and DL-based DDoS attack detection systems are discussed. The most relevant approaches to defend against such adversarial attacks are reviewed in Section 2.2.3. Finally, in Section 2.2.4, the limitations of these defence mechanisms are discussed to highlight our contributions.

2.1 Non adversarial input perturbations

ML methods have been proven to be vulnerable to non-adversarial input perturbations, such as additive Gaussian noise [11–13, 17]. In this thesis, we study the robustness of ML methods to non-adversarial perturbations in the context of ICS anomaly detection. An ICS is a representative example of dynamic systems operating in harsh and noisy environments, and therefore, the ADSs are susceptible to input perturbations, as explained in the following subsections.

2.1.1 Industrial Control Systems

ICSs exist in critical infrastructures (e.g., water treatment plants and power grids) where they control and monitor the functionality of industrial processes. The control process can be fully automated or include a human in the loop. A typical ICS consists of several components, including sensors, actuators and Programmable Logic Controllers (PLCs), which act together to achieve industrial objectives such as water treatment. These components interact using an array of industrial network protocols on a layered network architecture [18]. A sensor is a device that measures a physical quantity or phenomena and converts it proportionally to an electric voltage or current. By contrast, actuators generate a rotary motion or move linearly in response to an ingress power from a control device. This power can be electrical, hydraulic, or of another type. Actuators are then classified by their power type, such as electric or hydraulic actuators [19]. A control device can be a human, an electronic system or a PLC.

A PLC is a computational device, a single-purpose computer, with programmable memory to store user-oriented programs. The PLC executes user-oriented programs in infinite scan cycles, each of which contains three stages: obtaining inputs from sensors, executing the user program and

sending actuation commands to actuators. PLCs collect the readings from the sensors that monitor the status of the physical process and send the appropriate commands to the actuators. Based on the PLC's internal logic, such commands may be used to either change or maintain the current state of an actuator [18].

Another component of ICS is the Human-Machine Interface (HMI), which can be either software or hardware that allows operators to control and monitor the system from a central place.

ICSs are usually arranged in architectures such as PLC centred for small industrial processes or SCADA. SCADA systems are employed when control and data collection are equally important. Such systems collect data from geographically distributed locations in real time and send the data to a centralised computer that uses a SCADA software. The goal is to help operators control and monitor the process from a central location [19]. SCADA systems often include data historians that store the collected data. Such a data historian is a common server between the control network and the corporate network, allowing the flow of data between both networks. Corporate networks are telecommunication networks that use traditional network protocols such as TCP/IP to manage data. The flow of data between control and corporate networks has provided attackers with a route to reach the SCADA systems to carry out disruptive attacks on the controlled industrial process, as explained next.

Attacks targeting ICSs

Nefarious cyber-attacks targeting ICSs may cause service downtime or material loss in industrial production sites, with potentially negative repercussions on the lives of citizens. Notable examples are the cyberattacks against the Iranian nuclear program in 2010 and the Ukrainian power grid in late 2015.

The cyberattack against the Iranian nuclear program was perpetrated using the Stuxnet worm [20]. The worm was developed to be autonomous such that it copies itself from one memory stick to another until it reaches a specific ICS. This worm exploited several zero-day vulnerabilities, making itself undetectable by signature-based detection systems. After reaching the targeted SCADA system, the worm infected the PLCs and altered the code to damage the controlled centrifuges inside the plant by repeatedly alternating the rotation speed between high and low. Stuxnet proved that air-gaping control networks might not be sufficient to protect ICSs.

Another example is the cyberattack against the Ukrainian power grid in late 2015 [1]. In this multistage attack, the perpetrators started with a spear-phishing attack using BlackEnergy (BE) malware against employees of three regional power distribution companies. By doing so, they were able to gain access to corporate networks and, thus, the connected SCADA systems. Next, the attackers altered the firmware of specific control devices and instructed the breakers to perform an unscheduled malicious operations. Finally, they carried out a Denial of Service (DoS) attack on the call centre to prevent companies from updating consumers about the blackout. The attack affected thousands of consumers and left them without electricity.

The increasing occurrence of such attacks and their complexity has motivated the development of intrusion detection solutions for ICSs and the establishment of real-world ICS testbeds. In such testbeds, real cyberattacks are carried out to collect datasets with realistic attack scenarios. These datasets allow the research community to evaluate the proposed attack detection systems. Moreover, such testbeds provide a realistic environment where non-adversarial input perturbations exist. Non-adversarial input perturbations can appear as a Gaussian noise added to the device readings, causing frequent deviations in the ADS input and thus generating false positive alarms. It can also appear as a normal behaviour evolution that

adds perturbations to the input data, causing the input data to deviate from the learnt model of normality and thus causing more false positives.

In the next subsections, there is a brief introduction on the two ICS testbeds that have been used to collect two known datasets, as summarised in Table 2.1. Then, examples of natural input perturbations in both testbeds are provided.

The SWaT dataset and testbed

This dataset was collected from the SWaT testbed, a reduced version of an operational, clean water treatment plant [21, 22]. The water treatment process is monitored by a SCADA workstation and divided into six stages, each of which has a specific function. For example, stage one supplies raw water to the testbed, as shown in detail in Figure 2.1. This stage comprises a motorised valve MV101 that controls the water flow into a tank, a sensor FIT101 that measures the water flow, a sensor LIT101 that measures the water level in the tank, a pump P101 drives water to stage two, and a pump P102, which is a hot-standby backup pump for P101.

The six stages of the testbed are illustrated in Figure 2.2, which is adapted from [21]. Stage two is a pre-treatment process in which the quality of the raw water is analysed, and chemicals are blended with the water to improve the quality. Next, stage three employs consecutive filters in order to remove residuals from the water. Stage four is a dechlorination process that reduces the chlorine level to an acceptable limit. Stage five is responsible for removing inorganic residuals from the water before sending it to stage six, which stores the treated water for distribution.

Each stage is controlled by a pair of PLCs that communicate with sensors and actuators. All the PLCs are connected in a star-like network topology. On the top, there are HMI devices, a SCADA workstation, and a server to collect historical data.

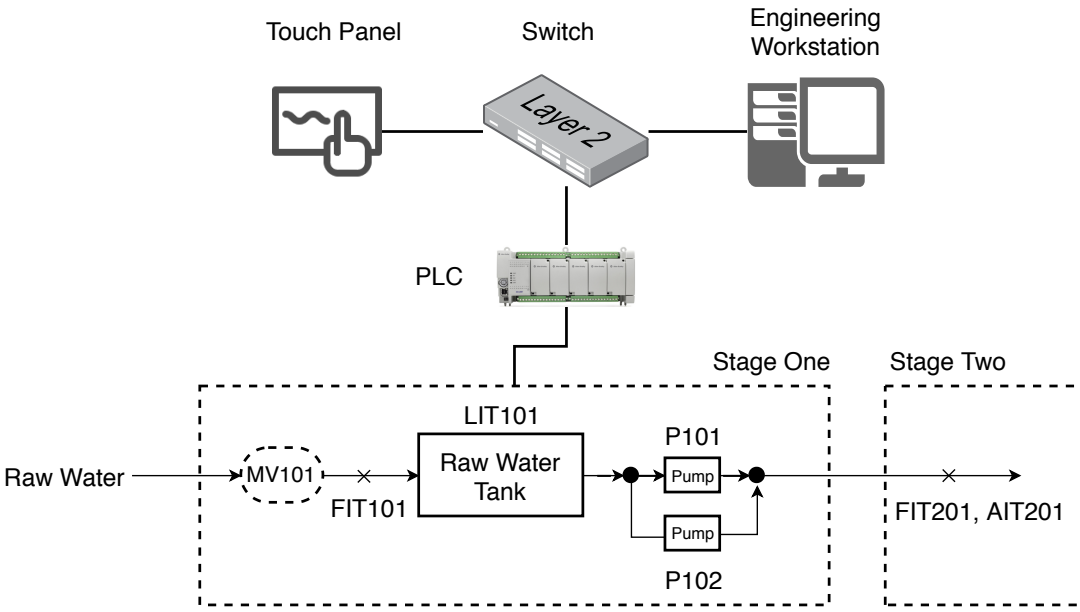

Figure 2.1: A simplified architecture of the first stage of the SWaT testbed. The arrows show the water flow inside the testbed.

The testbed contains 51 field devices divided into 25 sensors and 26 actuators. The 25 sensors in the testbed include Level Indication Transmitters (LIT), Flow Indicator Transmitters (FIT) and Analyser Indication Transmitters (AIT), while the 26 actuators include pumps (P) and motorised valves (MV). The actuators only take discrete states, i.e., each actuator is either *ON* or *OFF*, while sensors take on continuous states, for example, LIT101 has a measurement range of 0.2 to 6 meters.

The dataset consists of 946,722 records of sensors and actuators collected during 11 days of operation at a rate of one sample per second. Each record contains 51 attributes, representing 25 sensor readings and 26 actuator states. The dataset is divided into a 7-day portion of normal operations, which has been used as the training set, plus a 4-day portion of normal activity combined with 36 attacks generated by following the attack model in [23] (the test set); 20% of the training set was used for validation (validation set).

The attacks in the test set are of a duration ranging between two minutes

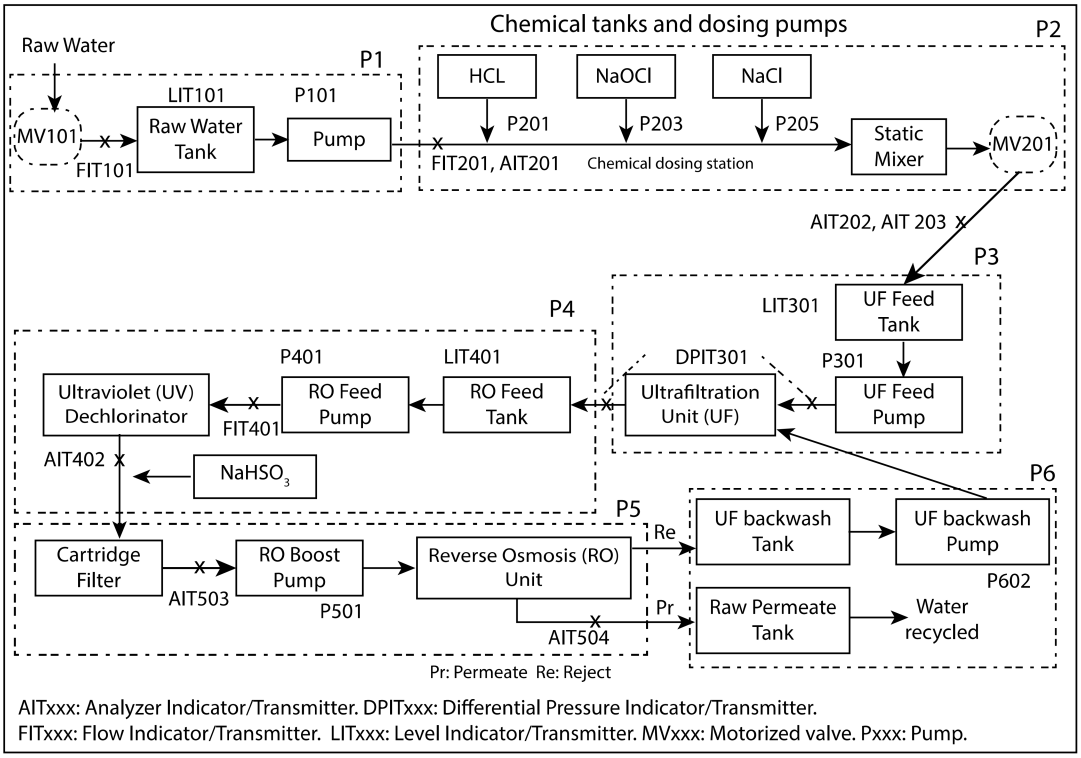

Figure 2.2: Overview of the SWaT process. The arrows show the water flow inside the testbed.

and nine hours and involve one or multiple devices at the same time. Attacks include spoofing the readings of sensors or reversing the operation states of actuators. For instance, in the first attack, the state of a motorised valve is switched from close to open for 15 min to cause a tank overflow. In another attack, the value reported by a water level sensor is decreased by 0.5mm/sec for 28 min, as depicted in Figure 2.3, again with the intention of causing a tank overflow. A more detailed description of these attacks is available in [21, 24].

The WADI dataset and testbed

The WADI testbed is a reduced version of a real-world water distribution network [22, 25], as depicted in Figure 2.4. The testbed network was designed to distribute 10 gallons/min of treated water. It consists of

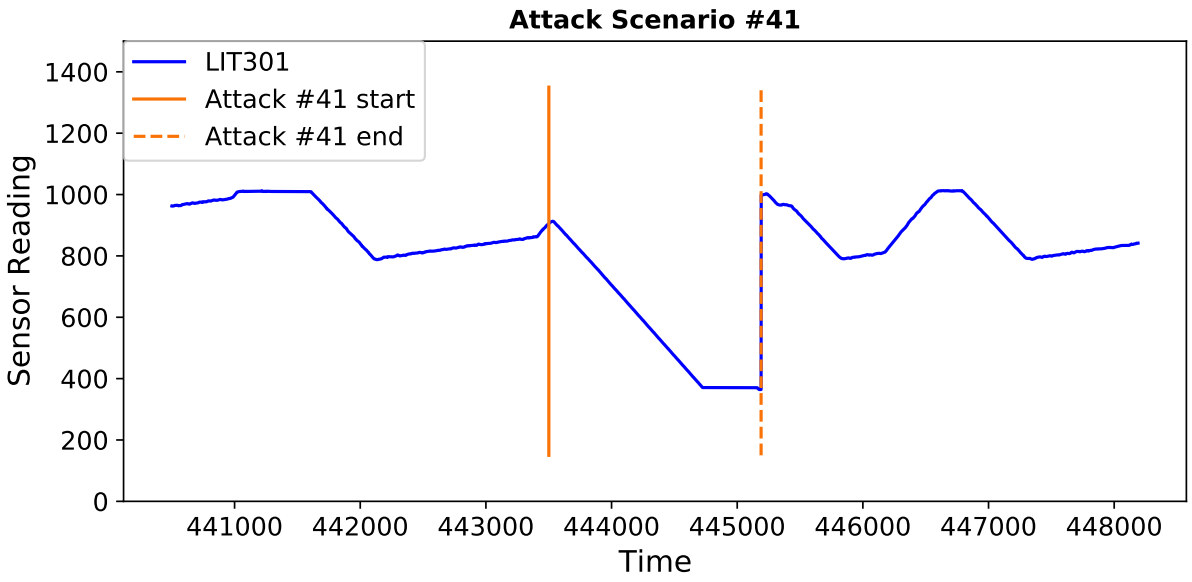

Figure 2.3: An illustration of an attack in the SWaT dataset. The attacker spoofed the reading of sensor LIT301 to decrease by 0.5mm/sec.

three sub-processes or stages, including a primary grid for water supply, a secondary grid for water distribution to six consumer tanks, and a return water grid that handles the excess of water from the consumer tanks.

The primary grid is controlled by one PLC and consists of two water tanks that can receive the filtered water from the SWaT testbed, plus chemical dosing components that can be activated to maintain the water quality. The secondary grid is controlled by three PLCs and consists of six consumer tanks in addition to two elevated reservoirs. The consumer tanks receive their water demand from the reservoirs according to certain patterns to simulate real consumer behaviour. Finally, the return water grid is managed by one PLC and consists of one tank and a chlorine neutralisation module [22].

Similar to the SWaT testbed, WADI is supervised by a SCADA workstation. Sensors include water flow indication transmitters, level indicator transmitters, analyser indicator transmitters and pressure meters. The actuators include pumps and motorised valves.

Figure 2.4: An illustration of the WADI testbed.

The WADI dataset comprises 1,209,610 records, each with 123 attributes divided into 69 sensor readings and 54 actuator states collected during a period of 16 days. Normal operation conditions were recorded during the first 14 days, split into training (95%) and validation (5%) sets. The last two days, with normal activity and 15 attacks, are used as the test set. The attacks follow the same model used in the SWaT dataset, which is described in [23], although with shorter durations (between 2 and 30 min). For instance, one attack cuts off the water supply to consumer tanks, and the other aims to increase the level of chemicals in the water by sending false sensor readings (sensor 1_FIT_001_PV) to the PLC for ten minutes while the sensor is actually turned off, as shown in Figure 2.5. More details and examples of WADI attacks can be found in [25].

Examples of non-adversarial perturbations

Non-adversarial input perturbations appear in the SWaT and WADI datasets as a normal behaviour evolution that perturbs the readings of control de-

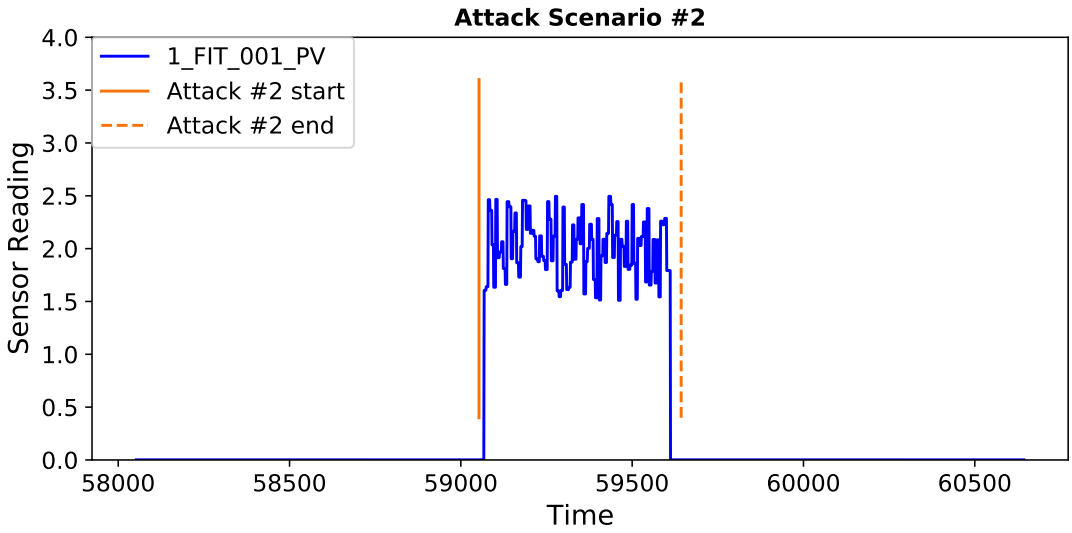

Figure 2.5: An illustration of an attack in WADI dataset. The target sensor was turned off before attack while the attacker is sending false readings to the PLC.

Table 2.1: Overview of the SWaT and WADI datasets

Dataset	Records	Duration (days)	Attacks	Sensors	Actuators
SWaT	946,722	11	36	25	26
WADI	1,209,610	16	15	69	54

vices. This normal behaviour evolution causes a distribution shift between the historical data in the training set and the unseen data in the test set. Indeed, the probability distribution of some sensors changes between the SWaT training and test sets. For example, in the training set the output of the Analyser Indication Transmitter $AIT201 \in [251, 272]$ $\mu S/cm$, while in the test set $AIT201 \in [168, 267]$ $\mu S/cm$ with a substantial different distribution, as illustrated in Figure 2.6.

The behaviour evolution problem can also be observed in the WADI dataset. For instance, the records of one of the *Flow Indication Transmitters* range within [0, 2.3] $m^3/hour$ of water in the training set, while the range changes to [0, 3.3] $m^3/hour$ in the test set. Further details of perturbations in both datasets are provided in Section 3.1.

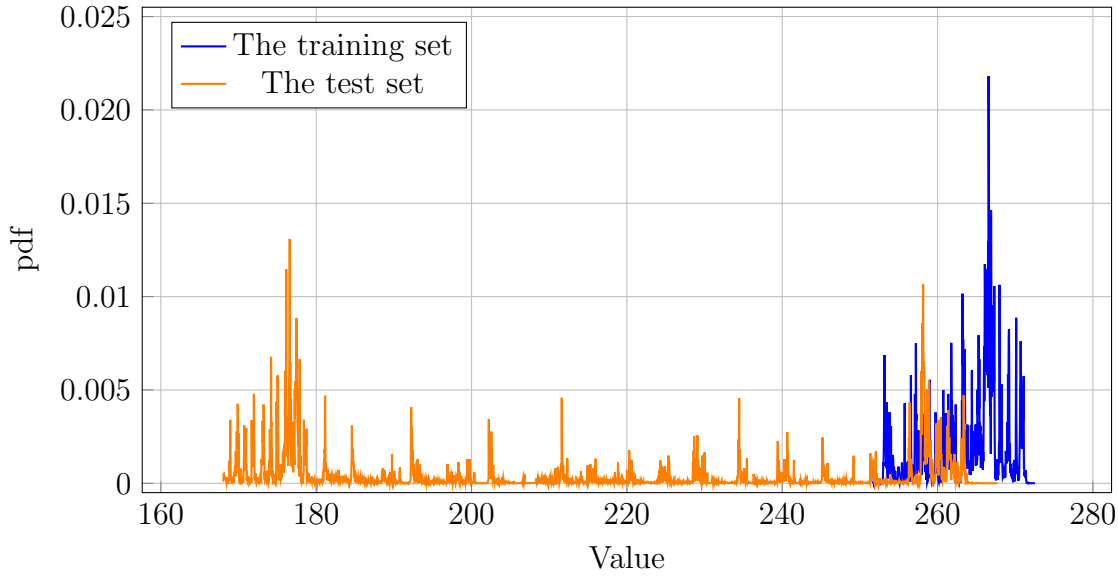

Figure 2.6: The probability density function for sensor AIT201. It provides an indication about changes in the normal behaviour between the training set and the test set.

The SWaT and WADI datasets are widely used by the security research community to evaluate anomaly detection solutions, as explained in the next section.

2.1.2 Anomaly detection in ICS

Detecting cyberattacks targeting ICSs depends on ADSs that raise the alarm when the industrial process deviates from its expected behaviour. ADSs build a model representing the normal behaviour by using ML and DL models that are trained with a normal dataset. In an online ADS, the model is employed to predict the normal behaviour of the ICS, and the prediction is then compared with the observed behaviour to calculate a residual. If the residual between the predicted and observed behaviours is higher than a predefined threshold, the observed behaviour is considered anomalous.

ML employs computational statistics and information theory to develop computer systems that learn to execute tasks through training [26]. ML

techniques employed in anomaly detection include Support Vector Machines (SVM) and Random Forest (RF).

SVM based classifiers were employed by Ahmed et al. [27, 28] to detect attacks in a clean water treatment plant (SWaT testbed). The classifiers analysed a set of time-domain and frequency-domain features, which were extracted after establishing the noise fingerprints of the sensors. A noise fingerprint combines the noise from the physical process, which can be duplicated, with the noise from manufacturing imperfections that are unique by nature. The a one-class SVM used in [27] gave favourable results in comparison with the multi-class SVM employed in [28].

A major limitation in [27,28] is that a full retraining of the SVM model is required in case the normal behaviour evolves (natural input perturbations) because the noise component from the physical process changes and hence the fingerprint. Overall, SVM classifiers are scalable and can perform real-time anomaly detection. Nevertheless, these classifiers have extensive memory requirements, and their performance relies on the choice of several key parameters, such as the kernel function and its related parameters.

RF is another ML technique employed to build ADSs in both power grids [29] and consumer IoT networks [30]. Wang et al. [29] employed an AdaBoost classifier [31] with RF as its base classifier to detect disturbances and cyber-attacks in power grids. Doshi et al. [30] used an RF classifier to detect DDoS traffic in consumer IoT networks. The authors used feature engineering to extract stateless and stateful features from network traffic. Stateless features include the packet size, protocol and inter-packet interval, while stateful features include bandwidth usage and frequency of change in destination IP addresses.

It has been shown that RF classifiers have a low computational footprint and are robust to overfitting [29, 30]. Nevertheless, RFs can be less efficient when trained on vast amounts of data, which is the usual situation in CIs,

as an increasing number of Decision Trees (DTs) need to be constructed.

As can be seen from the methods presented above, ML techniques depend on hand-crafted features to learn to execute a specific task, which limits their ability to learn directly from raw data. This limitation is addressed using a set of ML methods, called representation learning, which automatically extract and discover the features from the raw data to learn different tasks.

DL techniques are representation learning methods as they can automatically process raw data to determine the patterns and representations required for training to learn a task [32]. The ability to find representations with limited prior knowledge and without human intervention, combined with the availability of high computational power, have made DL methods suitable for learning directly from vast amounts of high-dimensional data [32]. The following anomaly detection methods use either DL models or an ensemble of ML and DL methods.

Sestito et al. [33] employed a Multilayer Perceptron (MLP) to detect three classes of anomalies while the normal operation they considered a fourth class. The MLP was trained on a dataset collected from the network traffic of a manufacturing plant. In another study, Shalyga et al. [34] used an MLP to detect anomalies in the SWaT testbed, but the false-positive rate was high. To solve this problem, they used a weighted p-powered error calculation in addition to a smoothing moving average that was exponentially weighted. The framework in [34] detected 25 out of 36 attacks in the dataset.

However, MLP is known to have a considerable number of trainable parameters, and therefore long training and test durations [35]. Consequently, the applicability of MLPs in ADSs can be questioned in scenarios where anomaly detection is performed in real time. This drawback has motivated the use of Convolutional Neural Networks (CNNs), which are a variant of MLPs with fewer trainable parameters and low computational

requirements [35].

There are examples of ADSs based on CNN classifiers in [36,37]. Zheng et al. [36] proposed an ADS to detect electricity theft in smart grids. They used convolutional layers arranged in a wide and deep architecture [38] to learn the periodicity of normal electrical consumption and the irregular behaviour of electricity theft. The neural network was trained in a supervised manner with a binary decision function representing either a normal consumption or a theft. The authors in [37] trained a 1D-CNN model with unsupervised learning to predict the future readings of sensors and actuators of a water treatment plant with anomalies detected when the difference between the predicted and actual readings exceeded a predefined threshold. The models in [36, 37] have the advantages of a short training time and high detection accuracy.

Long Short-Term Recurrent Neural Networks (LSTM-RNNs) are another class of neural networks that have the ability to learn temporal relations, long-term as well as short-term interactions from sequential data [39]. Goh et al. [40] presented an approach that uses an LSTM-RNN and a Cumulative Sum (CuSum) technique to detect anomalies. The CuSum technique sets the upper and lower control limits for the prediction error in each sensor and actuator. An anomaly is detected when the prediction error is outside limits. In addition to the high false-positive rate, the main limitation of this approach is that it requires expert human intervention to tune and update the CuSum limits after a normal behaviour evolution. Moreover, LSTM-RNNs-based ADSs have a large number of trainable parameters that increase the training and inference times, reducing their applicability in fields with high-speed processing requirements such as anomaly detection.

A Deep Auto-Encoder (DAE) is another type of neural network that is used to build ADSs [41, 42]. AL-Hawawreh et al. [41] trained a DAE and then replaced the decoder with fully connected layers for anomaly

classification. The model was trained with supervised learning. In [42], Kravchik and Shabtai trained a DAE using features extracted from both the time and frequency domain readings of control devices. The results obtained by using the frequency-domain features showed that they are unsuitable for detecting short-duration anomalies because some dominant frequencies require a long time window for analysis. Notably, Kravchik and Shabtai [42] have reported the problem of normal behaviour evolution in the SWaT testbed. They employed the Kolmogorov-Smirnov (K-S) test to compare the probability distributions of each control device across the training and test datasets. However, devices with behaviour evolution were excluded from the modelling to reduce the number of false alarms. In a practical scenario, excluding devices could pose a serious threat to ICS.

GANs are also used in network security. For example, Li et al. [43] presented an ADS based on a GAN. They used one model for the entire ICS to capture the latent relationships between all sensors and actuators. In this case, the GAN was built using Recurrent Neural Networks (RNNs), which require long training and test periods.

Another research line has combined ML and DL methods to benefit from their strengths. Abokifa et al. [44] proposed an ensemble model that incorporated four techniques, namely, an MLP, statistical fences, Principle Component Analysis (PCA) and a module verifying the states of actuators. The first three techniques attempt to detect different types of anomalies, whereas the last module detects violations of the physical process rules. Interestingly, the authors discussed the need for retraining the models in the case of behaviour evolution, and they pointed out that statistical fences and the PCA are easy to retrain, unlike MLP. Retraining MLP could be problematic because of the required historical data.

The approach presented in [45] is based on computing the distance between the current and previous device states without prior knowledge. It

is assumed that this distance should be below the threshold during normal operations. The proposed solution was tested using the SWaT dataset. Another study proposed the TABOR ADS [46], which was validated on the SWaT dataset. TABOR combined different models, namely: Probabilistic Deterministic Finite Automaton (PDFA) and a Bayesian Network (BN). The final anomaly detection was based on a combination of the results obtained from these models. In this case, the authors do not address the problem of updating the model in case of normal behaviour evolution. Updating the model here appears to be cumbersome because of the complex characterisation of the interaction between the sensors and actuators needed to build the model.

The literature on anomaly detection in ICSs is extensive. Survey papers covering the recent advances in this research area include [47, 48].

Discussion

The research reviewed in this section shows that anomaly detection is well covered in the literature. However, the problem of non-adversarial input perturbations is overlooked, despite its role in increasing the number of false positives produced by ADSs [4]. The existing literature requires full retraining of the underlying DL and ML models as well as hyperparameter tuning.

For instance, [40] suffers from a high false-positive rate and requires expert human intervention to tune and update the CuSum limits. Another example is [37], which uses a CNN and a statistical approach for anomaly detection based on a normalised value of the prediction error. The normalisation is computed using the statistical properties of the prediction error of the benign samples recorded during normal operations. However, the authors do not explain how such statistical properties are updated in the case of changes in normal behaviour or noisy data readings. In addition, in the case of

TABOR [46], the authors do not address the problem of updating the model, a task that appears cumbersome because of the complex characterisation of the interaction between sensors and actuators needed to build the ensemble model.

In summary, a common drawback of available solutions is that they are not flexible enough to adapt quickly and efficiently to changes in the production environment. In a water treatment plant, examples of such changes are the deployment of a new water tank, which influences the behaviour of the sensors attached to it, or the replacement of a motorised valve with another with different operation modes.

ADSs rely on a threshold to specify the maximum allowed prediction error during normal operations, such that above this threshold a system state is considered anomalous. A low threshold may increase the number of false alarms, while a high threshold may allow anomalies below the threshold to pass undetected. Therefore, the proper selection of threshold can make the ADS robust to natural input perturbations and decrease the number of false alarms. The threshold is either selected empirically by an expert [37] or based on the prediction error of the training set [45, 49]. In [50, 51], Markov chains and support vector regression were employed to dynamically tune thresholds for anomaly detection in network traffic (e.g., port scans, brute force attacks, SQL injections, etc.) and robot-assisted feeding. To the best of our knowledge, automated threshold tuning has not yet been implemented in ADSs for ICSs.

Another important aspect of ICS is the additive Gaussian noise that emerges naturally in ICSs because of device ageing and electromagnetic interference. Such additive noise causes the field devices (i.e., sensors and actuators) to report noisy readings, increasing the false positive rate and providing a chance for attackers to hide their attacks [52]. Therefore, it is challenging to detect anomalies in noisy data, and such a challenge is

overlooked in the ADS literature.

2.1.3 Comparison with this thesis

Through this literature review, we identified a research gap that can be addressed by answering four research questions:

- How can the underlying DL model adapt to the non-adversarial input perturbations in ICS?
- How can the anomaly detection threshold be automatically tuned to reduce false positives and alleviate the burden on the human operator?
- Can ADSs that are robust to noisy data readings be developed?
- Can the similarity between alarms be used to reduce the false alarms reported to the operator?

The DAICS ADS presented in Chapter 3 addresses the first three research questions. DAICS comprises an algorithm called *few-time-steps* to update the model of normality according to the changes in the normal behaviour of the ICS. This algorithm uses a small amount of data to update the neural network weights and requires minimal to low expertise human intervention. DAICS also includes an automated threshold tuning technique based on a simple neural network to provide DAICS with the ability to cope with the dynamic behaviour of industrial environments. The sensitivity of DAICS to noisy data was measured and it was demonstrated that DAICS can maintain a baseline of protection even when data readings are highly corrupted by noise.

The last research question is addressed by the SiFA framework presented in Chapter 4. SiFA collects a buffer of historical false alarms and suppresses every new alarm similar to these false alarms. This buffer is updated (a new instance is added or an old instance is replaced) whenever the system

operator identifies a new false alarm. Moreover, when the DAICS anomaly detection logic detects an alarm, the similarity between this alarm and every instance in the buffer is computed using the squared Euclidean distance. The computed minimum distance is compared with a similarity threshold. When the computed minimum distance is above the threshold, this alarm is considered dissimilar to the historical false alarms, and is hence reported to the operator. Below this threshold, the alarm is considered to be similar to at least one historical false alarm, and hence, suppressed. SiFA is validated using the SWaT dataset, and the results show that SiFA can reduce the false alarm rate of DAICS by a factor of five.

2.2 Adversarial input perturbations

ML models have been shown to be vulnerable to adversarial input perturbations, which are perturbations carefully selected by attacker to force the ML model to provide a specific output [53, 54]. In this thesis, we study the problem of adversarial robustness in the context of ML-based NIDSs detecting DDoS attacks, which are increasingly common in recent years [55]. The application domain of DDoS attack detection constrains the process of generating adversarial traffic as such traffic must maintain the malicious behaviour and remain valid according to the network protocols. In the following, we present a brief introduction to the DDoS attack, ML-based NIDSs, and adversarial attacks against such detection systems.

2.2.1 DDoS attacks

A DoS attack is an effort to prevent legitimate users from accessing an online service by either consuming all the available bandwidth or by exhausting the target system resources. A successful DoS attack causes a service slowdown or a server crash after using all the available resources. DoS attacks carried

out by multiple and distributed sources are DDoS [56].

Over recent years, a common method of carrying out DDoS attacks is by attackers gaining access to a large number of scattered computing devices, forming a botnet. The botnet is remotely controlled to send a massive volume of malformed network traffic to a victim server or network. The goal is to exhaust the bandwidth (i.e., network/transport-level flooding attacks) or consume the victim resources, including CPU and memory (i.e., application-level flooding attacks) [57].

In this thesis, the focus is on two volumetric DDoS attacks, namely, TCP-SYN and HTTP-GET flood attacks. The TCP-SYN flood attack is an example of a network/transport-level flooding attack in which attackers exploit a feature of the victim protocol to consume the server's resources [57]. The attackers send a huge amount of TCP Synchronise (SYN) requests to the victim, pretending to be legitimate users initiating network connections. The victim acknowledges each SYN request by sending an SYN-ACK packet, leaves an open port and waits for those clients to acknowledge the communication. The clients' acknowledgements never arrive as the attackers often spoof the source IP in the SYN requests [56].

HTTP-GET attack is another type of volumetric DDoS attack in which the attackers generate a high rate of HTTP-GET requests, overwhelming the victim server. This is not a spoofed attack as the attackers have to establish connections to send the flood of valid GET requests. However, attackers do not wait for a response from the victim [56]. The result of such a volume of valid HTTP requests is a high number of open connections in the server, exhausting the server resources (e.g., CPU and memory). One or more botnets can be used to carry out the attack in order to open connections and send as many HTTP-GET requests as possible, preventing a legitimate user from opening a connection.

DDoS attack detection

The complexity and volume of DDoS attacks continue to increase, causing significant disruption to critical online services and leading to economic losses. For instance, the number of DDoS attacks is expected to reach 15.4 million globally by 2023 [58]. Meanwhile, a new record of 2.3 Tbps was attained in an attack targeting Amazon Web Services (AWS) in Q1 2020 [59]. To achieve this scale of damage, attackers develop more sophisticated attacks by combining multiple vectors (i.e. protocols) and adapting their attacks to bypass NIDSs.

In recent years, NIDSs have evolved to leverage ML models to cope with the detection of new attacks, which cannot be identified by signature-based detection methods. An ML-based NIDS essentially consists of a feature extractor and a ML-based model, as depicted in Figure 2.7. The feature extractor processes the raw network traffic to derive features that are arranged into data samples suitable for input to the model. The model is trained using malicious and benign samples and then deployed to classify new samples extracted from live traffic. Several ML-based NIDSs have been proposed and demonstrated detection of DDoS attacks with high accuracy [60–62].

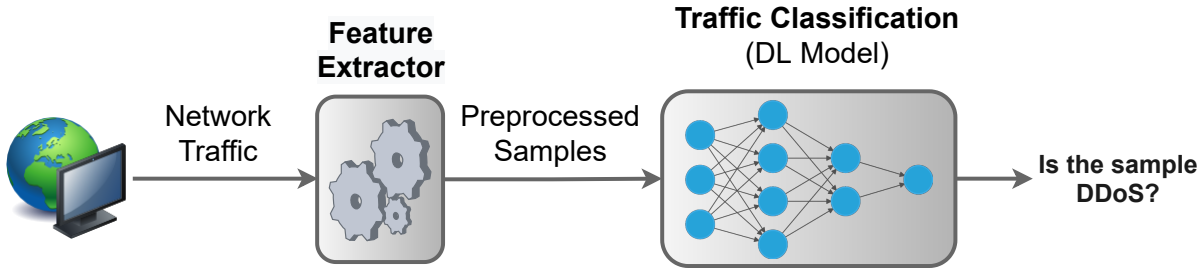

Figure 2.7: An illustration of an ML-based NIDS. The feature extractor (i.e., preprocessing tool) receives the network traffic to then extract data samples suitable for classification by the DL model. The output is a decision classifying the sample as belonging to a DDoS attack or not.

In [61], Doriguzzi et al. presented Lightweight, Usable CNN in DDoS

Detection (LUCID), which is an efficient DDoS detection system that implements a pre-processing tool for the network traffic and a CNN. The preprocessing tool, referred to as a feature extractor, converts the flows extracted from the network traffic into data samples compatible with the CNN. The CNN classifies the data samples as either benign or DDoS. Each data sample includes a set of features extracted from each packet of a bidirectional flow in a specific time window. LUCID was evaluated on several datasets including CIC2017 and CSECIC2018 [63] with an average F1 score $> 99\%$. More details on LUCID are introduced in Chapter 5.

Elsayed et al. [62] proposed DDoSNet, which is a DDoS attack detection framework based on a combination of RNN neural networks and autoencoders. To prepare the network traffic for classification, the authors used the CICFlowMeter [64] as a preprocessing tool to extract more than 80 features from each network flow. The autoencoder consists of eight RNN layers that encode the input feature vector to have a smaller dimension before decoding again to approximately the same input. This combination of RNN and autoencoders allows for better DDoS attack detection. DDoSNet was evaluated on the CICDDoS2019 dataset [65] and demonstrated a detection accuracy above 99%.

Kitsune [60] is another state-of-the-art NIDS that is based on an ensemble of autoencoder neural networks. Each autoencoder has a single compression layer, which makes it very efficient for training and execution. Kitsune relies on a feature extractor in order to obtain more than 100 statistical features from each network flow. These features are then mapped into a set of compressed representations that can be used for anomaly detection by an ensemble of autoencoders. The NIDS was evaluated on several attack scenarios including traffic from a Mirai botnet and an SYN flood attack. The results show a true positive rate of above 99% in most of the evaluated attacks.

The ML and DL models, however, have been proven to be vulnerable to adversarial attacks [66]. An adversarial attack is an imperceptible perturbation added to the input, leading to a change in the model's classification decision. In the same way, ML-based NIDSs are vulnerable to adversarial attacks. The adversarial attack aims to evade the classification model and reach the target server/network to achieve the intended malicious goal. In a DDoS scenario, the attacker slightly perturbs the characteristics of the malicious network flows such that the feature extractor gives samples misclassified as benign. Such an attack scenario has been demonstrated to be realistic in previous studies [8–10].

In the next section, the recent advances in the evaluation of ML-based NIDSs in an adversarial setting are reviewed.

2.2.2 Adversarial attacks against NIDSs

There are two approaches to implementing adversarial attacks against NIDS; perturbing data samples and perturbing network traffic[1]. In the first approach, the adversary introduces perturbations directly on the preprocessed samples before feeding them to the ML model. In the second approach, the adversary generates adversarially perturbed DDoS flows, which are then processed by the feature extractor to create data samples for the ML model to classify. This approach can be more realistic than perturbing extracted samples that require the adversary to bypass the feature extractor to feed such samples into the model.

The vulnerability of learning-based NIDSs to adversarial perturbation has been explored and implemented using the aforementioned two approaches, namely, perturbing data samples and perturbing network traffic.

[1]Formally, this equates to the feature space and the problem space, respectively, as studied in [67].

Perturbing extracted samples

The key to the success of an adversarial evasion attack is to maintain the characteristics of the attack. For example, for a SYN flood attack that sends a large number of SYN packets (i.e. SYN flag set), a feature based on the *flags* of the TCP header should not be manipulated. The attack approaches presented in [68–70] do not consider this. They introduce adversarial perturbations in the features of a sample without excluding the features, the manipulations of which might alter the attack. As a result, these approaches are impractical in real-world scenarios.

Yan et al. [71] split features in the KDDCup99 dataset into *unchangeable* features that are essential to maintain the malicious function and *changeable* features that can be manipulated without affecting this function. Then, the authors employed a Wasserstein GAN (WGAN)-based neural network to manipulate the changeable features of the DDoS attack to evade a CNN-based NIDS. The reported results show a 50% drop in the detection accuracy of the NIDS on the KDDCup99 dataset.

Hashemi et al. [72] propose an adversarial attack that manipulates the changeable features of malicious traffic based on three perturbations: splitting the payload to increase the number of packets in a flow, modifying the delay between subsequent packets sent from the attacker and injecting decoy packets among the real attack packets. These perturbations were used to manipulate 11 attacks from the CICIDS2017 dataset [63] while targeting different NIDSs, including a modified version of Kitsune [60]. The results demonstrate that the evaluated NIDSs are vulnerable to adversarial attacks. For instance, Kitsune experienced an approximately 40% drop in the detection accuracy of the DDoS attack and an average drop of approximately 30% across all attacks.

However, both works [71, 72] lack a discussion on how these perturbed

samples can be reproduced in live network traffic. This live traffic should be valid according to the different network protocols in order to pass through the network and carry out the intended attack function on the victim side.

Perturbing network traces

Kuppa et al. [10] proposed an attack approach suited for NIDSs that use threshold-based anomaly scores to classify network flows. The adversaries try to find the minimal input distortion that is sufficient to change the system decision. The authors identify features such as the inter-arrival time where distortion can be added in both directions without compromising the malicious functionality of an attack. This approach was applied to perturb the traces of the CICIDS2018 dataset [63] in order to target NIDSs based on ML models. The results show that the attack success rate, i.e., the ratio of misclassified samples to all adversarial samples, was above 50% across all models. Moreover, the NIDSs considering both the reconstruction error and feature distances for anomaly scoring are more robust.

Sadeghzadeh et al. [9] presented an approach to generate adversarial network traffic based on three different adversarial attacks, namely, AdvPad, AdvPay and AdvBurst. The AdvPad attack injects perturbations either at the beginning or end of the packet payload to evade the packet-based NIDSs, which take as input the byte sequence of a packet and outputs a label for each packet. The AdvPay attack injects perturbations in the payload of dummy packets that are sent at a random location of a network flow. This attack aims to fool the flow content-based NIDSs, which take as input the byte sequence of the first n packets of a flow and each flow is labelled. Finally, the AdvBurst attack sends a number of dummy packets with manipulated statistical features at the end of a network flow to fool the flow time series classifiers, which label each network flow based on the statistical features of the first m packets, e.g., packet inter-arrival time.

The attacks were used to deceive network classifiers that aim to discover the traffic type, e.g., chat, email or streaming. All classifiers were based 1D-CNN models. The results suggest that the AdvPay is more effective, if the perturbation is added to the beginning of the payload. Furthermore, the AdvPay and AdvBurst attacks add low bandwidth overhead to the network traffic and significantly decrease the recall of the classifiers.

Aiken et al. [8] investigated adversarial attacks targeting an anomaly based NIDS within Software-Defined Networks (SDN). The authors proposed perturbing three features: payload size, packet rate and volume of bidirectional traffic, which can be implemented in live traffic without compromising the attack function. The attack is implemented in SYN flood traces that are classified by several ML models, including SVM and K-Nearest Neighbors (KNN). The results show that by combining perturbed features, a detection accuracy drop from 100% to 0% is observed across some of the classifiers and the KNN classifier is more robust than the others. The results also demonstrate that an attacker can perturb the characteristics of malicious network flows while maintaining the harmful functionality.

2.2.3 Defence mechanisms in NIDSs

In the following, we review some of the related works proposed to increase the robustness of NIDSs, which are based on denoising autoencoders [73], adversarial training [74, 75], ensemble voting [76] and combining adversarial training with ensemble voting [77].

Hashemi et al. [73] proposed two solutions to increase the robustness through denoising the samples to remove adversarial perturbations. The first solution is called Reconstruction from Partial Observation (RePO) in which a random mask hides a part of an input sample that will be fully reconstructed by a denoising autoencoder. The usage of a random mask

might hide the perturbed part of the sample, and hence the autoencoder reconstructs a clean copy for anomaly detection. The second solution is called RePO+ and uses 100 masks, instead of one mask in RePO, which has been empirically proven to be more efficient in removing the adversarial perturbations. The authors evaluated their approach on the CICIDS2017 dataset using the adversarial attacks presented in [72]. The results suggest that RePO+ is more robust than other NIDSs such as Kitsune [60].

Abou Khamis et al. [74, 75] proposed a defence mechanism based on the min-max optimisation approach [78]. The Min-max optimisation aims to simultaneously *(i)* find the best adversarial samples that maximise the loss of the target classifier, i.e., the samples are misclassified, and *(ii)* employ these adversarial samples to train the classifier in order to minimise the loss, i.e., the classifier is more robust. These samples were crafted using four gradient-based attack methods [79]. The results show that using the rFGSM^k, which is an extension of the Fast Gradient Sign Method (FGSM) algorithm, to generate the adversarial training samples leads to robust models [75]. In [74], the authors extend the approach in [75] to evaluate different architectures of neural networks, including CNN and RNN, on two datasets, namely, the UNSW-NB15 [80] and NSL-KDD [81]. The results suggest that the models trained on the UNSW-NB15 dataset are more robust in comparison with those trained on the NSL-KDD dataset.

Shu et al. [76] proposed a defence approach called Omni based on ensemble models. The ensemble is created using the so-called unexpected models, which are models with hyperparameters very different from those of the attacker's target model. This approach assumes that an attacker uses an optimal model (i.e., expected model) to design the adversarial attack, and thus, such an attack is ineffective against sub-optimal models (i.e., unexpected models). The optimal model is found using hyperparameter optimisation to give an optimal evaluation performance. On the other

hand, the sub-optimal model provides a sub-optimal evaluation performance and has an architecture that is dissimilar to that of the optimal model. The architecture dissimilarity between the optimal and sub-optimal models makes the adversarial examples nontransferable between both models. Furthermore, the use of a weighted ensemble of sub-optimal models forces the attacker to design adversarial perturbations that can evade all models, which can be a complicated task. The approach was evaluated on several network security datasets, and the results show that Omni can build ensemble models robust to evasion attacks.

Zhang et al. [77] proposed a defence approach that combines three techniques: ensemble voting based on three different ML models, ensemble adversarial training [82], and query detection. A flow is classified as benign only if all the models classify it as benign, forcing the adversary to craft samples that can evade all models simultaneously, which might be a complicated task. Moreover, they use ensemble adversarial training to augment the training dataset with adversarial samples, thereby improving the robustness of individual models. The query detection technique thwarts the attacker's attempt to obtain a considerable number of (sample, label) pairs to improve the attack. The defence approach is evaluated on the CICIDS2018 dataset [63] and was effective in reducing the success rate of adversarial attacks.

Discussion

Despite the good performance of these defence approaches, they are only evaluated on perturbations that are implemented on extracted samples. For instance, Hashemi et al. [72, 73] executed the adversarial attacks on samples extracted from network flows using the CICFlowMeter [64]. Abou Khamis et al. [74, 75] used samples from the preprocessed dataset files to carry out the adversarial attacks. In [76, 77], the perturbations were implemented

on samples taken from the preprocessed dataset files. However, there is no discussion on how to convert such extracted samples into real network traffic, as network traffic is convertible to extracted samples but not vice versa. Furthermore, the threat model of these approaches assumes that the adversary can access the input of the classification model to feed the perturbed samples, which can be unrealistic as such a model is preceded by a feature extractor in real world.

Another limitation is the use of adversarial attacks unsuitable for volumetric attacks, e.g., HTTP flood DDoS attacks, wherein a high volume of traffic is sent to the victim server within a short period. For instance, Zhang et al [77] generated adversarial samples for evaluation using five black-box attacks. However, these attack approaches require at least 150 iterations (i.e., queries) to adjust the perturbations added to each malicious sample to evade the detection model. In a practical scenario, this means a tremendous amount of queries is required to carry out a successful volumetric attack, allowing the victim to discover these queries, blacklist the attacker's IPs and thus prevent the adversarial attack.

A common limitation of adversarial training approaches [74, 75, 83] is the use of adversarial samples that are generated based on the target/victim models. For example, in [74, 75], the victim model is trained on adversarial samples generated using the same model in a white box setting. In such approaches, the victim model learns to generate weak adversarial samples, instead of learning to defend against strong perturbations and hence remains vulnerable to adversarial attacks [82].

2.2.4 Comparison with this thesis

This literature review has helped us identify a research gap that can be addressed by answering three research questions:

- Can the process of generating adversarial samples for training be decoupled from the target model?
- Can robust NIDSs be developed and evaluated under practical conditions?
- Can a potential overestimation of robustness be identified?

In Chapter 5, the first two questions are answered by proposing an adversarial training approach called GADoT, in which the target model is decoupled from the generation of the adversarial samples used for training. The adversarial samples are generated by perturbing DDoS samples with values obtained from a GAN. Thus, the trained model learns to defend against strong perturbations and can be resistant to adversarial attacks.

For a practical evaluation, the approach of perturbing the network traffic was adopted to generate two datasets in which the semantics of DDoS attacks are fully preserved. The first dataset contains six traces with adversarial SYN flood attacks [84], while the other is based on the CICIDS2017 dataset [63] and contains traces of adversarial HTTP GET flood attacks [85]. These traces capture perturbations that are suitable for volumetric DDoS attacks. The evaluation results on both datasets are the basis of our conclusions regarding the performance of GADoT. To the best of our knowledge, GADoT is the first solution to be evaluated against representative real-world network attack scenarios.

In Chapter 6, the third research question is addressed by proposing a universal adversarial attack called UPAS to identify a potential overestimation of robustness. UPAS is considered a black-box attack because it requires no knowledge of the ML classification model, training dataset, or detection logic and packets' content. The UPAS attack injects a random time delay between 0 and 10 seconds before packets from attacker, making the attack suitable for perturbing live network traffic. Furthermore, this

attack adds no bandwidth overhead. For a practical evaluation, the attack activity is injected in the malicious traces of a multi-attack dataset and used to benchmark the robustness of two NIDSs, which are based on a denoising autoencoder and an adversarially trained DL model.

Chapter 3

Robustness to non-adversarial input perturbations

The upsurge in cyber-attacks targeting CIs may cause service downtime or material loss, with potential negative consequences for the lives of citizens. Because of the increasing occurrence and complexity of such attacks, ML-based ADSs for CIs have been developed. One-class classification is a popular and powerful technique for developing ADSs. In the training stage, solutions based on one-class classification build a model that represents the normal behaviour of the CI (the class of "normality"). In the production stage, the detection system uses this model to verify whether the behaviour of the live system matches the expected normal behaviour. Deviations from normality are usually determined through thresholds on the classification errors and flagged as anomalies.

ML-based ADSs have the advantage of being sensitive to abnormal behaviours, which can include zero-day attacks and faulty devices. However, such ADSs can also be sensitive to non-adversarial input perturbations, which increase the number of false positive alarms. In this thesis, we investigate the problem of vulnerability to non-adversarial input perturbations in the case of an ML-based ADS monitoring ICS. Real-world ICSs are dynamic and evolve over time, which may "shift" the normal behaviour,

perturbing the input of the ADS, thus causing false alarms and hampering the correct functioning of the detection system. ICSs also operate in noisy environments, which may add noise to sensor readings, causing frequent deviations in normal behaviour, thus generating additional false alarms. In such scenarios, the following two aspects are particularly challenging: (i) updating the model of normality upon changes in ICS behaviour, and (ii) adjusting the thresholds according to the collected data. Previous studies either do not address such issues [42, 46] or require the intervention of human experts to adjust the model parameters [40, 49], as explained in Chapter 2.

Based on the aforementioned challenges, three research questions are identified:

- How can the underlying ML model adapt to the normal behaviour evolution in ICS?
- How can the anomaly detection threshold be automatically tuned to reduce false positives and the burden on human operators?
- Can ADSs that are robust to noisy data readings be developed?

To address these questions, in this chapter, we propose a framework called Deep learning Anomaly detection in Industrial Control Systems, namely DAICS [14, 15]. DAICS is an anomaly detection solution for ICSs based on a one-class classification paradigm. This solution combines a database-oriented management approach and deep learning architecture to model the normal behaviour of the ICS. The proposed neural network relies on *wide and deep* learning and *convolutional layers* to memorise and generalise the characteristics of the ICS. Wide and deep learning is a technique that has been recently proposed to improve the performance of recommender systems [38]. DAICS implements an automatic threshold tuning algorithm and the so-called *few-time-steps* algorithm, which is based

on the few-shot learning paradigm [86, 87], to quickly update the model with the latest changes in the ICS normal behaviour.

We evaluate DAICS using datasets collected from the SWaT and WADI testbeds, which are widely used for security research on ICSs [22]. The datasets comprise a training set containing only normal records and a test set with normal and anomalous records. The anomalies in the test sets are real-world attacks that target the integrity and availability of the testbeds [23]. DAICS was compared with other solutions in terms of anomaly detection accuracy and robustness to noisy data.

The rest of the chapter is structured as follows: the problem statement is introduced in Section 3.1. Section 3.2 defines the threat model used in this research. The proposed anomaly detection framework is explained in Section 3.3. Experimental setup and the evaluation results are presented in Section 3.4. We summarise this chapter in section 3.5.

3.1 Problem Statement

Unsupervised anomaly detection solutions for ICSs usually rely on the so-called *one-class classification* technique. The basic idea is to build a model of the normal behaviour of the industrial process and to consider as anomalous every event that does not fit the model. The main challenge with such approaches is dealing with the *non-adversarial input perturbations* (can be also called dataset shift[1]), which appear as a gap in the data distribution between the training and unseen data [69, 88]. A direct consequence of such perturbations is an increase of false alarms generated by the anomaly detection system due to normal events classified as anomalies.

The non-adversarial input perturbations problem is present in the SWaT dataset, where we can observe changes in the normal behaviour of some

[1]This equates to a distribution shift due to process variations, as described in [52]

devices across the training and test sets. For instance, during the normal operation, the pump P102 has a single state of value 1 in the training set, then it takes an additional "normal" state of value 2 in the test set. Also, the probability distribution of some sensors changes between the two sets. For example, in the training set the output of the Analyser Indication Transmitter $AIT402 \in [153, 236]\ mV$, while in the test set $AIT201 \in [141, 328]\ mV$ with a different distribution, as illustrated in Figure 3.1.

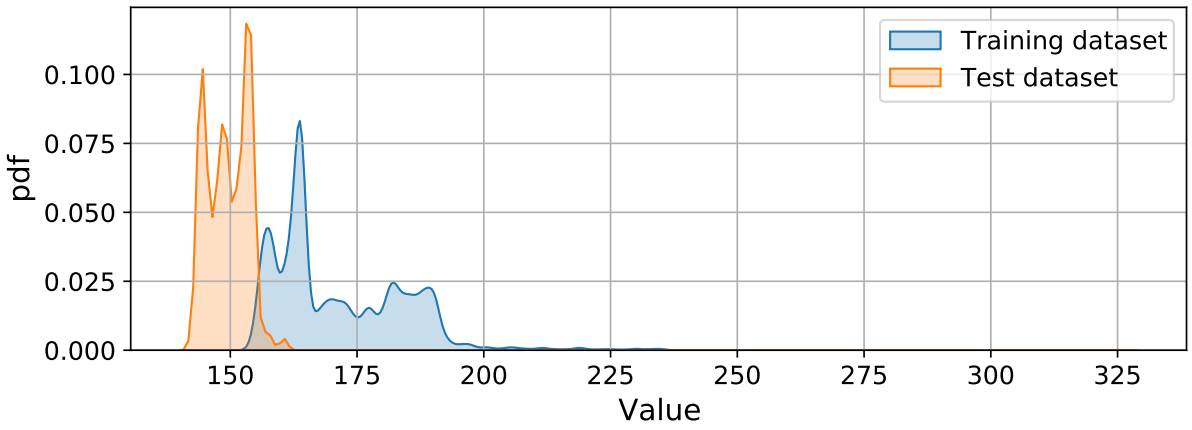

Figure 3.1: The probability density function for sensor AIT402 of the SWaT testbed. It shows the variation in the normal behaviour between training and test sets.

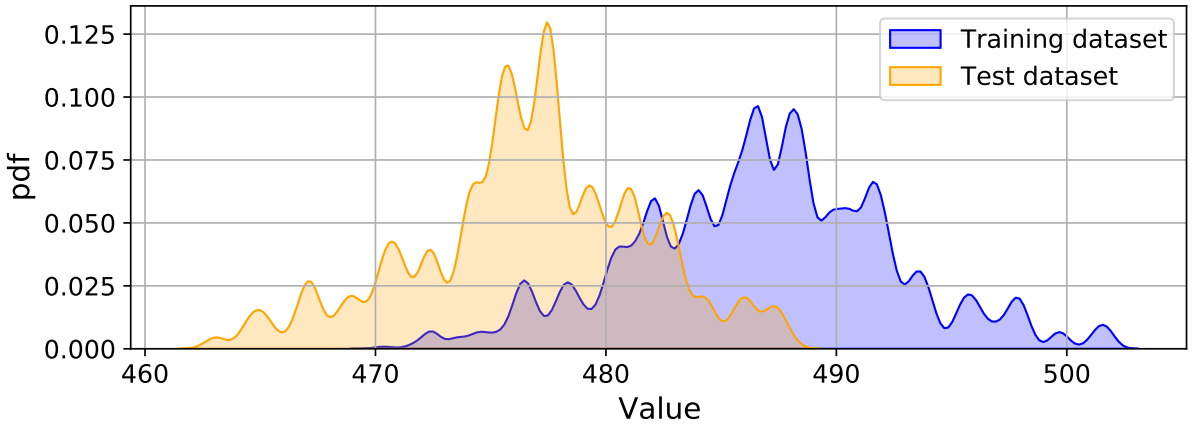

Figure 3.2: The probability density function for sensor $2A\_AIT\_004$ of the WADI testbed. It shows the variation in the normal behaviour between training and test sets.

Another relevant example is the presence of redundant devices, such as the redundant pump P102 in the SWaT testbed. A redundant pump is always off until the primary pump stops working for any reasons (hardware or software faults, etc.). In such situations, the PLC turns on the redundant pump to take over the work of the primary pump. If this process is not covered in the training set, the forecasting model will consider the operations of the redundant pump as anomalies.

The non-adversarial input perturbations problem can also be observed in the WADI dataset. For instance, the records of the Analyser Indication Transmitters $2A\_AIT\_004$ range within [470, 501] mV in the training set, while the range changes to [462, 487] mV in the test set, as shown in Figure 3.2. The exact motivation of the non-adversarial perturbations problem in the two datasets is not described in the documentation. However, as these behavioural changes are not part of the datasets' threat models, and as we observe them only in normal operating conditions, we assume they are not caused by malicious actions.

Modelling of the normal behavioural evolvement is still a challenge for the implementation of the anomaly detection solutions in real-world settings [89,90]. Existing approaches have tackled the perturbations problem by only focusing on adjusting the parameters of the detection algorithms, or by excluding from the modelling process those devices that present the behaviour evolution problem, as discussed earlier in Chapter 2. However, we argue that adjusting the detection parameters without updating the model of normality is not sufficient to cope with dynamic environments such as ICSs.

In the next sections, we describe the threat model of DAICS and the approach to tackling the *non-adversarial input perturbations* problem. The main idea is to follow the changes in the industrial process by automatically updating the model of normality and the anomaly threshold.

3.2 Threat Model

We assume that the attacker's capabilities include gaining remote access to the networked control system, to the SCADA workstation, or can physically compromise sensors and actuators within the ICS. We also assume that the attacker has domain knowledge of the targeted ICS, e.g., the physical property measured by each sensor and the physical consequences of actuation commands. The attacker's goal is to use the above capabilities and knowledge to damage or modify the ICS operations. This includes: (i) altering the state of actuators through a MITM-like attack, in which the actuator receives commands from the attacker instead of a PLC; (ii) sending spoofed sensors readings to the PLC to drive the PLC to take wrong decisions; and (iii) tampering with the PLC firmware aimed to take the ICS out of service or to change the programmed logic (e.g., the Stuxnet worm [20]). However, the attacker does not have enough knowledge of the ADS to perform adversarial attacks, i.e., to evade the detection logic by introducing noise in the system or by generating crafted data that keeps the anomaly metric below the detection threshold [91–93].

3.3 The DAICS framework

DAICS builds the model of normality by combining a neural network with a database-like approach. Deep learning is used to model the continuous readings of sensors, whose values are sampled periodically by a PLC, while the database stores the states of actuators controlled by the PLC. Moreover, DAICS implements the few-time-steps learning algorithm to update the model based on the evolution of the ICS. Anomaly detection involves comparing the actual behaviour of the system against the model. Deviations beyond a dynamic threshold are considered as anomalies. A glossary of the

Table 3.1: Glossary of symbols

WDNN	Wide and Deep Neural Network
TTNN	Threshold Tuning Neural Network
W_{in}	Input time window
W_{out}	Output time window
W_{anom}	Anomaly waiting time
H	Horizon
G	Number of output section of WDNN
$\mu_{out}[g]$	Output section g of WDNN
m_{ac}	Total number of actuators
m_{se}	Total number of sensors
m_{se}^{g}	Number of sensors of output section g
$\text{MSE}_{g,t}$	Prediction error on section g at time t
$\text{MSE}_{g,[a,b)}$	List of prediction errors on section g between time a and $b-1$
T_g	Anomaly threshold for section g
ν_t	Actuator states recorded at time t

notation used in this chapter is presented in Table 3.1.

3.3.1 Prediction of sensor states

The normal behaviour of sensors is modelled using a deep neural network named Wide and Deep Neural Network (WDNN), which predicts the normal states of sensors based on the past states of both sensors and actuators. The idea of wide and deep neural networks was introduced in [38] to build a recommender system that suggests apps based on the user's query and preferences. Our WDNN, shown in Figure 3.3, has two goals: memorisation through the wide branch and generalisation through the deep branch. Memorisation means learning the relationship between feature-pairs in the training set, hence recording the co-occurrence of combinations of sensors values. Generalisation means the ability to explore relationships that do not exist in the training set.

The neural network comprises a *Feature extractor* section that learns

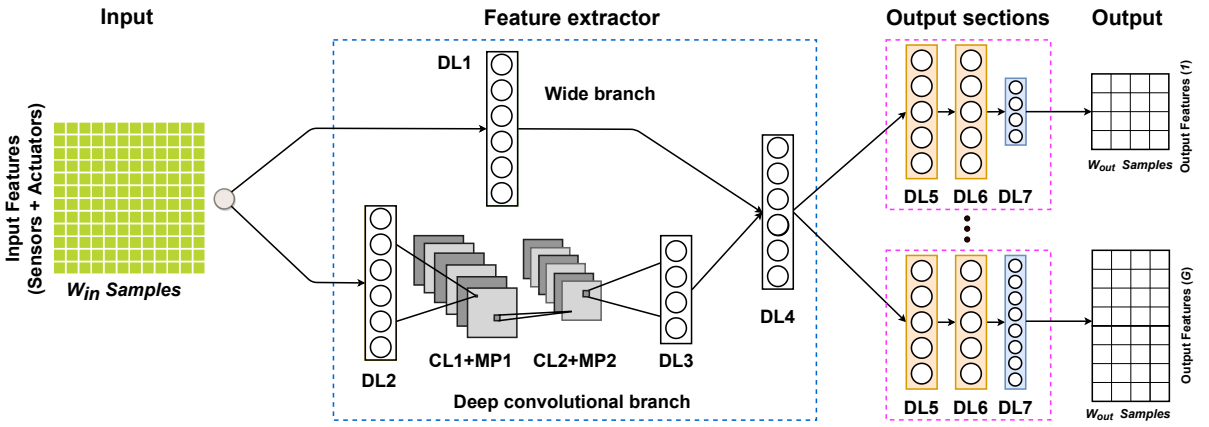

Figure 3.3: Architecture of WDNN. The prediction of the sensors states is computed combining the states of both sensors and actuators. The size of DL7 varies according to the number of output features of the output section.

the relations between all sensors and actuators during the normal operation of an ICS and extracts the features required to predict the normal states of sensors. Moreover, the output section is split into multiple branches to serve large-scale ICSs where the sensors are controlled by several PLCs, usually specialised on a specific part of the industrial process with specific behaviour and anomaly threshold.

Architecture. The neural network takes as input an array X of data samples collected during a time window of length W_{in} seconds, corresponding to W_{in} samples, as the dataset was collected at a sampling rate of one sample per second. The size of X is $m \times W_{in}$, where $m = m_{se} + m_{ac}$ is the number of features for each sample including the state of m_{se} sensors and m_{ac} actuators taken at time t. Observe that we use the states of both sensors and actuators because the future states of sensors depends on their current states and the actions taken by actuators. The output Y is the expected normal states of sensors during a future time window W_{out}. Both time-windows W_{in} and W_{out} are separated by a time interval called horizon H. The purpose of H is to prevent the neural network from replicating the last values of the input time window W_{in} into W_{out}, as pointed out by Shalyga

et al. [49].

As shown in Figure 3.3, the *Feature extractor* section comprises two convolutional layers and four fully connected layers organised into two branches. The fully connected layer DL1 represents the *wide branch*, which learns the normal interactions between sensors and actuators using cross-product transformations between the input features.

The *deep branch* allows the WDNN to generalise to events not covered in the training set, hence to reduce the prediction error in the case of new normal combinations of input states. This branch comprises the fully connected layer (DL2), two one-dimensional convolutional layers (CL1 and CL2), each one followed by a max-pooling layer (MP1 and MP2), and the fully connected layer (DL3). As shown in other works (e.g., [37]), one-dimensional CNNs are particularly suitable for modelling time series data. The purpose of layers CL1 and CL2 is to model the data collected from sensors and actuators in a specific time window. With max-pooling, we down-sample the output of each convolutional layer by a factor of 2. Finally, the fully connected layer DL3 re-shapes the output of MP2 to allow its concatenation with the output of the wide branch.

Both branches are aggregated in another fully connected layer (DL4) followed by multiple dense output sections, each one consisting of the three fully connected layers DL5, DL6 and DL7. Each output section learns the most relevant information from the aggregation layer in order to predict the normal behaviour of a group of sensors controlled by a specific PLC. Each fully connected layer can be described as $Y = LeakyReLU(\mathbf{W}^T X + \mathbf{b})$, where Y is the output, X input, $\mathbf{W}$ is an array of weights the model learns during the training, and $\mathbf{b}$ is the bias. We introduce non-linearity in the model by using the leaky rectified linear activation function defined as follows: $LeakyReLU(x) = max(0, x) + 0.01 * min(0, x)$. Similarly, the convolutional layers can be described as $Y_i = LeakyReLU(Conv(X, \mathbf{W}_i, \mathbf{b}_i))$,

where Y_i is the output of the convolution on the input X using the i-th filter with weights $\mathbf{W}_i$ and bias $\mathbf{b}_i$.

Cost function. The neural network presented above has been trained to find the set of weights and biases that minimises the Mean Square Error (MSE) cost function. The cost function computes the error between the model predictions on sensors readings and the corresponding observed values. Hence, by minimising the cost, we reduce the prediction error. At the training stage, the cost function for a batch size of s samples (i.e., s different time windows) in a specific output section g can be formally written as:

$$c_g = \frac{1}{s}\sum_{t=1}^{s}\Big(\frac{1}{m_{se}^g}\sum_{i=1}^{m_{se}^g}(Y_t[i] - \tilde{Y}_t[i])^2\Big) \tag{3.1}$$

where $Y_t[i]$ is the predicted value for sensor i at sample t (i.e. time-step t), while $\tilde{Y}_t[i]$ is the corresponding observed sensor value (the value present in the training set). m_{se}^g is the number of sensors in an output section g. The final cost c is the sum of costs of all G output sections $c = \sum_{g=1}^{G} c_g$, which is minimised using Stochastic Gradient Descend (SGD) [94].

3.3.2 Anomaly detection in actuators

The SWaT and WADI testbeds include discrete actuators such as pumps and motorised valves. Actuators in the SWaT and WADI testbeds include pumps and motorised valves. The pumps are arranged in pairs of primary and redundant hot-standby pumps. A redundant pump is turned on only in the case the respective primary pump stops working. This operational mode complicates building a forecasting model that predicts the actuator states, as some actuators (such as the redundant pumps), are rarely used during the normal operations. As a consequence, after measuring a high prediction error with DL-based methods due to lack of normal operation

records, we designed a light and straightforward approach based on querying a database containing all the normal actuator states. A database entry is n-tuple, where n is the number of actuators in the testbed ($n = 26$ in the SWaT, $n = 54$ in the WADI). Each entry is a combination of actuators states labelled as normal, for a total of 146 entries available in the SWaT dataset and 2001 entries in the WADI dataset. An example of tuple from the SWaT testbed is provided in Table 3.2.

Table 3.2: Tuple in the database of actuators normal states

Actuator	MV101	P101	P102	...	P601	P602	P603
Tuple	2	2	1	...	1	1	1

At testing time, the combinations in the test set with no occurrences in the training set are marked as anomalies, as explained in Section 3.3.3.

As demonstrated in Section 3.4, this approach works well with benchmark datasets such as SWaT and WADI, where only discrete actuators are present. Nevertheless, real-world ICSs may also comprise actuators with continuous output values (e.g., continuous valve actuators [95]). Due to the huge number of possible output combinations, a database-oriented does not seem very practical for such deployments. With enough training data, the output of continuous actuators can be predicted with WDNN, as done for the continuous sensors states. Of course, this requires the extension of one or more WDNN output sections to support the actuators states, depending on the adopted ICS partitioning scheme (as also discussed in Section 3.3.1 for the prediction of sensor states).

3.3.3 Detection logic

DAICS operates on batches of actuators and sensors values retrieved from the PLCs at regular time intervals of duration s seconds. For the sake of simplicity, we assume that s also corresponds to the number of readings in

one batch for each device (like in the two datasets used in the experiments). We therefore define $I = [\bar{t}, \bar{t} + s)$ the time interval under analysis (where $[a, b)$ indicates the interval between a and $b - 1$). Given the observation time $t \in I$, DAICS verifies whether the current combination of actuator values is present in the database A built at the training stage. An anomaly is reported otherwise.

In the case of the sensors, the values observed at time t are compared against those predicted by WDNN using past values of sensors and actuators, as previously explained in Section 3.3.1. More precisely, the values observed at time $t \in I$ are compared with the first element of the output of WDNN obtained from values of sensors and actuators collected in the time window $[t', t' + W_{in})$ (one input sample for WDNN), where $t' = t - H - W_{in}$. The MSE value of each WDNN output section is evaluated against an adaptive threshold to determine the presence of any anomalies.

The detection process on one single output section g of the neural network is summarised in Figure 3.4, which also depicts an adaptive threshold mechanism for automatically tuning the anomaly threshold based on the past prediction error. Such a mechanism is described in Section 3.3.4. The region of the timeline highlighted in light blue represents the time interval I described above.

More precisely, the samples observed on the group g of sensors at time t are identified as anomalous when the anomaly condition $\text{MSE}_{g,t} > T_g$ is met, where T_g is a threshold on the prediction error for the group of sensors g. Using the same notations as for the cost function in Equation 3.1, we define $\text{MSE}_{g,t}$ as follows:

$$\text{MSE}_{g,t} = \frac{1}{m^g_{se}} \sum_{i=1}^{m^g_{se}} (Y_t[i] - \tilde{Y}_t[i])^2 \tag{3.2}$$

To reduce the false positives caused by sudden changes in the underlying

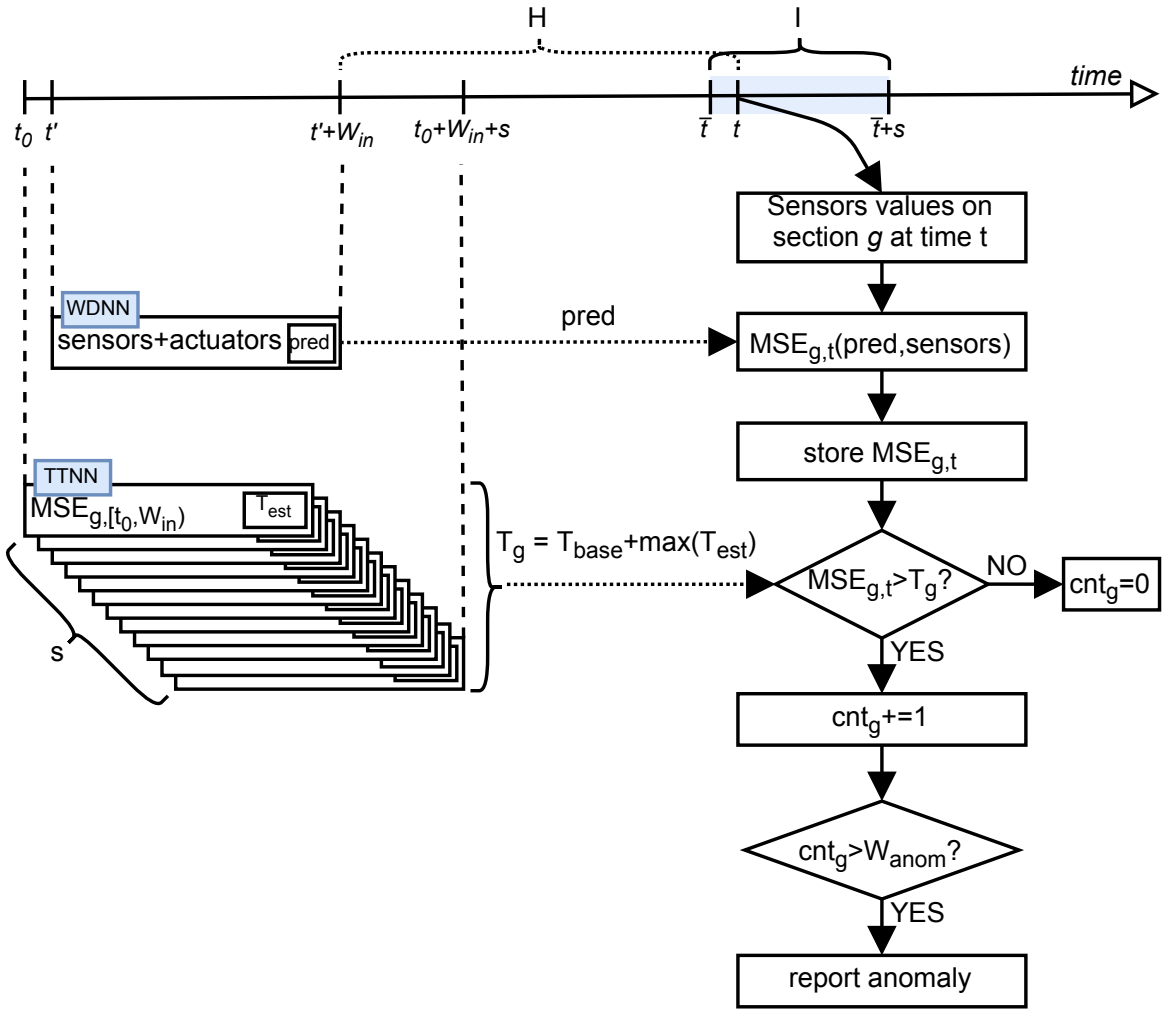

Figure 3.4: Detection process on section g of sensors through WDNN and adaptive thresholds.

physical process (as also observed by Kravchik et al. [37]), DAICS reports an anomaly at time t only if the anomaly condition has also been previously observed for W_{anom} consecutive sampling intervals. In Figure 3.4, this process is represented through increasing the counter cnt_g.

In summary, the anomaly condition can be expressed as:

$$L_t = \begin{cases} 1, & \text{if } \exists g \in [1, G] \text{ s.t.} \\ & \text{MSE}_{g,i} > T_g \ \forall \ i \in [t - W_{anom}, t] \\ 1, & \text{if } \nu_t \notin A \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

where ν_t is the combination of the actuators values observed at time t, while $L_t = 1$ defines the anomaly condition at time t. W_{anom} is one of the hyper-parameters used to tune DAICS.

3.3.4 Threshold tuning

The industrial processes are dynamic and their behaviour change over time, hence hampering the accuracy of anomaly detection systems. In this regard, a major challenge for such systems is finding the threshold to classify an event either as an anomaly or as normal activity. Existing approaches tackle this problem through empirically, by adjusting the threshold at test time as a hyper-parameter [37, 40, 49]. While empirical thresholds produce good results in the laboratory with static datasets such as SWaT and WADI, production systems can hardly afford long threshold tuning sessions upon new noise levels or novel operation modes. Instead, here we propose an adaptive technique to dynamically tune the threshold based on the prediction error trend.

Ali et al. [50] used a machine learning technique to estimate the threshold based on the historical prediction error of their ADS. In this research, we treat the prediction error MSE as a univariate time series that is modelled using a neural network that we called Threshold Tuning Neural Network (TTNN), whose architecture is depicted in Figure 3.5. We use G instances of TTNN, one for each output section of the WDNN model, to tune the

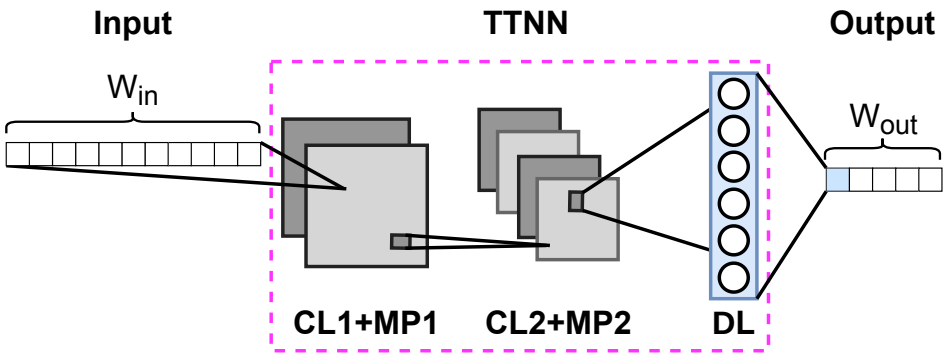

Figure 3.5: Architecture of TTNN. Only the first element of the output is kept. The size of the 1-dim kernel of CL1 and CL2 is 2 with stride 1.

thresholds $T_g,\ g \in [1, G]$. Each instance is trained with the prediction error MSE_g measured on a single output section g on the validation set, which only contains benign records, as the training set. The trained model is used in the online system to compute the optimal anomaly threshold T_g using past prediction errors. As represented in Figure 3.4, the threshold T_g used for anomaly detection on sensors values collected in time interval $[\bar{t}, \bar{t}+s)$ is obtained using past prediction errors computed in time interval $[t_0, t_0+s+W_{in})$, where $t_0 = \bar{t} - H - W_{in}$.

As shown in Figure 3.5, TTNN consists of two 1-dimensional convolutional layers, each followed by a max pooling layer, and of one final fully connected classification layer, represented in the figure as CL1, MP1, CL2, MP2 and DL respectively. The input is a time series of prediction errors measured for the samples collected in past time window of length W_{in} seconds, the output contributes to the estimation of the optimal anomaly threshold for the current sensors states. A detailed description of the threshold tuning process is described in Algorithm 1.

Referring to the timeline depicted in Figure 3.4, the anomaly threshold used at time $t \in [\bar{t}, \bar{t}+s)$ for output section g is computed by using a batch E_g of s samples defined as time series of past prediction errors of length W_{in}, starting from $\text{MSE}_{g,[t_0,t_0+W_{in})}$, $\text{MSE}_{g,[t_0+1,t_0+W_{in}+1)}$, to $\text{MSE}_{g,[t_0+s,t_0+W_{in}+s)}$.

Algorithm 1 Threshold tuning algorithm

Input: Batch of past prediction errors on section g ($E_g = \{\text{MSE}_{g,[t_0,t_0+W_{in})},...,\text{MSE}_{g,[t_0+s,t_0+W_{in}+s)}\}$), where $t_0 = \bar{t} - H - W_{in}$
Output: Anomaly threshold for WDNN output section g (T_g)
1: $T_{est} = []$; ▷ List of estimated thresholds
2: TTNN[g] ← load the weights and biases for section g;
3: $\bar{E}_g \leftarrow \text{median}(E_g)$ ▷ Median filter on the input
4: **for** $X = \bar{\text{MSE}}_{g,[t_0+i,t_0+W_{in}+i)} \in \bar{E}_g$ s.t. $i \in [0, s)$ **do**
5: $\hat{Y} = \text{TTNN}[g](X)$; ▷ Estimated prediction error
6: $T_{est}.append(\hat{Y}[0])$;
7: **end for**
8: $T_g = T_{base} + max(T_{est})$;

At line 3 of the algorithm, a median filter is applied to the input batch of past prediction errors E_g to reduce the impact of short-term changes of the prediction error on the threshold tuning process. Median filtering is accomplished by sliding a window over E_g while computing the median value of the elements of E_g under the window. The resulting median values are stored in array $\bar{E}_g$. One variable to consider in this process is the size of the median filter. We experimented with values between 50 and 120. As reported in Table 3.5, we obtained the best results with the median filter size of 59. In the loop at lines 4-7, the algorithm computes the estimated prediction error using the median values in $\bar{E}_g$. As shown at line 6, only the first element of the output layer is memorised for further processing. The threshold T_g is computed at line 8 as the maximum of the predicted thresholds plus the offset T_{base}. T_{base} is set as the sum of the mean plus the standard deviation of the prediction error on the validation set, and is kept constant after the deployment. This offset is added to reduce the sensitivity of the WDNN to small variations on sensors values. As an example, Figure 3.6 shows how the trend of threshold T_5 (the threshold for the output section number 5) follows the prediction error $\text{MSE}_{5,i}$ without going below it, except in presence of anomalies or malicious activities.

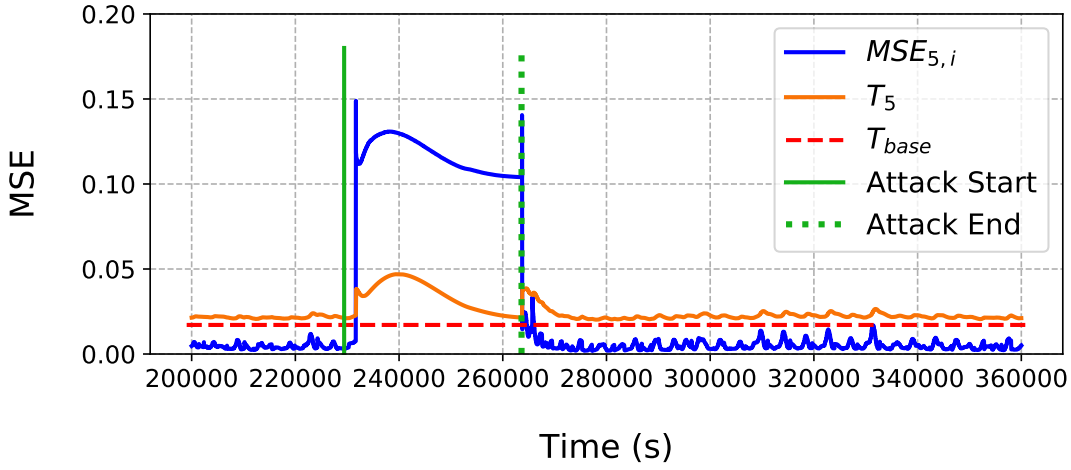

Figure 3.6: T_g and $\text{MSE}_{g,i}$ of the 5th WDNN's output section.

As a final step, the algorithm executes one SGD epoch to update weights and biases of each of the G instances of TTNN using the respective input batch E_g.

3.3.5 The few-time-steps algorithm

The few-time-steps algorithm has been designed to efficiently reconfigure DAICS in a production environment in the case an anomaly is identified by the system and then recognised as a false alarm by the technician.

In such a situation, DAICS is triggered by the technician to update the database A of actuators states and to tune the output section of the pre-trained WDNN. The only assumption is that the technician can recognise false alarms caused by changes in the normal operating condition of the ICS (e.g., changes in the hardware/software configurations). Unlike state-of-the-art solutions, the proposed approach only assumes a domain-specific understanding of the ICS operations, without requiring a deep knowledge of the algorithms and their thresholds and hyper-parameters.

Of course, the number of human interventions needed to handle false alarms is an important parameter to determine the usability of the anomaly

detection system. If the system is too sensitive, the technician would be overwhelmed by a large number of alarms to be verified, making it hard to spot those that are due to normal changes in the ICS behaviour, hence false. On the contrary, a conservative approach might not reveal true malicious activities or anomalies. Both cases can render the system ineffective, if not unusable. The sensitivity of DAICS to this parameter will be analysed in Section 3.4.

Algorithm 2 Few-time-steps learning algorithm

Input: Batch of samples (S), Technician's input: False alarm at time t ($FA_t[i]$) $i \in [0, G]$
Output: Retrained output section ($\mu_{out}[g]$)

1: **if** $\nu_t \notin A$ **and** $FA_t[0] ==$ **True then**
2: $A = A \cup \nu_t$; ▷ Update the database of actuators
3: **end if**
4: $EP_{tuning} \leftarrow$ number of fine-tuning epochs;
5: **for** $g \in [1, G]$ **do**
6: **if** $FA_t[g] ==$ **True then**
7: $\mu_{out}[g] \leftarrow$ weights and biases
8: **for** epoch **in** EP_{tuning} **do**
9: $c_g \leftarrow$ cost of section g on S;
10: $SGD(\mu_{out}[g])$ step to minimise c_g;
11: $\mu_{out}[g] \leftarrow$ update weights and biases;
12: **end for**
13: **end if**
14: **end for**

Algorithm 2 is called upon a technician's decision that an alarm detected by the system at time t is false. This means that Algorithm 2 is called when both of the following conditions are satisfied: the anomaly condition $L_t = 1$ and at least one of the elements of the Boolean vector $FA_t[i]$ ($i \in [0, G]$) is *True*. FA_t indicates where the false alarm has been detected, either among the actuators ($FA_t[0] = True$), or in one or more groups of sensors ($FA_t[i] = True,\ i \in [1, G]$), or both.

In the first case, with $FA_t[0] = True$, the algorithm adds the combination of actuator states ν_t to database A (lines 1-3), as the technician has

determined that ν_t is normal and must be treated as such by the system. In lines 5-14, the output sections of the neural network that have produced the false alarm are updated. Specifically, in lines 8-12, SGD is employed to fine-tune the output section through multiple gradient steps. We calculate the prediction loss for the data samples aggregated in a *batch* of S samples containing the false alarm (S is passed as an input to the algorithm). The optimiser minimises this loss by tuning the parameters of the output layers DL, DL6, and DL7. After EP_{tuning} optimisation steps (around 400 ms on average for 100 epochs on our testing environment described in Section 3.4.1), the updated output section $\mu_{out}[g]$ replaces the previous one in the anomaly detection process (line 11). Note that, grouping the sensors into different sections speeds up the execution of the few time steps algorithm, since only a portion of the output section is updated in the case of false alarms.

3.4 Experimental evaluation

In this section, we present a detailed evaluation of DAICS obtained using the SWaT and WADI datasets presented in Chapter 2. The evaluation comprises a comparison with state-of-the-art solutions in terms of detection accuracy, number of detected attacks and robustness to noisy data.

3.4.1 Experimental setup

DAICS has been implemented in PyTorch 1.0 [96] and validated using a Singularity container [97] running in a shared machine configured with 16 CPU cores, 64 GB virtual RAM and an NVIDIA 1080Ti GPU. The database of actuators normal states A is implemented as a NumPy array populated using the training datasets. The maximum size of the array is less than 1MB for each of the two datasets. Prior to our experiments, we

also normalised the sensor readings between 0 and 1.

3.4.2 Methodology

As per convention in the literature, we configure and evaluate DAICS using the following metrics:

$$Pr = \frac{TP}{TP + FP} \quad Re = \frac{TP}{TP + FN} \quad F1 = 2 \cdot \frac{Pr \cdot Re}{Pr + Re}$$

where *Pr=Precision*, *Re=Recall*, *F1=F1 Score*, *TP=True Positives*, *FP=False Positives*, *FN=False Negatives*. It is important to highlight that we adopt a point-based approach to compute these metrics. Thus, precision, recall and F1 score are computed using the labelled records (points) of the datasets, which report whether a record (a combination of actuators states and sensors readings recorded at a given time) is normal or anomalous. Like most of related works in the literature, we adopt this approach to tune the hyper-parameters of WDNN and TTNN models, to assess the overall performance of DAICS and for comparison with the state-of-the-art.

We used a grid search strategy to explore the set of hyper-parameters, including time windows W_{in}, W_{out} and W_{anom} in Table 3.3. The learning rate, batch size, number of layers in the output sections, number of neurons in each layer, and others listed in Tables 3.4 and 3.5. In the tables we report the final values that maximise the F1 score on the two datasets. These hyper-parameters are kept constant throughout our experiments presented below in this section, except when we study the sensitivity of DAICS to a specific hyper-parameter.

There were several paths to build DAICS including the use of a single output section for all sensors and applying a static threshold to the prediction error. However, the use of a single output section was not suitable for large ICS and caused performance degradation. Indeed, the AADS

framework [14], which uses a single output section, has F1 score lower than DAICS in both datasets, as shown later in tables 3.6 and 3.8. In the case of applying static thresholds, they reduced the precision of the anomaly detection process and therefore were not practical due to the evolving nature of the ICS environment.

Table 3.3: DAICS Hyperparameters

Hyperparameter	**SWaT**	**WADI**
Horizon (H)	50	20
Input time window (W_{in})	60	50
Output time window (W_{out})	4	4
Anomaly time window (W_{anom})	30	30

Table 3.4: WDNN hyperparameters

Hyperparameter	**SWaT**	**WADI**
Learning rate (α)	0.01	0.001
Batch size (s)	32	32
Tuning epochs (EP_{tuning})	100	100
Output sections layers	3	3
DL1 Neurons	W_{out}	W_{out}
DL2 Neurons	$3 * W_{in}$	$3 * (m_{se} + m_{ac})$
DL3 Neurons	W_{out}	W_{out}
DL4 Neurons	80	80
DL5 Neurons	2.25*m^g_{se}	2.25*m^g_{se}
DL6 Neurons	1.5*m^g_{se}	1.5*m^g_{se}
DL7 Neurons	m^g_{se}	m^g_{se}
CL1 Kernels, Kernel size	64, 2	64, 5
MP1 Pooling size	(64, 2)	(64, 2)
CL2 Kernels, Kernel size	128, 2	128, 5
MP2 Pooling size	(128, 2)	(128, 2)

In our experiments, we evaluate the sensitivity if DAICS to noise. In fact, ICSs operate on harsh industrial environments [98, 99], where the communication channels are often subject to interference (e.g., in the case of employing wireless communication devices [99]). For evaluation purposes,

Table 3.5: TTNN hyperparameters

Hyperparameter	**SWaT**	**WADI**
Learning rate (α)	0.01	0.01
Batch size (s)	32	32
Tuning epochs (EP_{tuning})	1	1
Median kernel size	59	59
CL1 Kernels, Kernel size	2, 2	2, 2
MP1 Pooling size	(2, 2)	(2, 2)
CL2 Kernels, Kernel size	W_{out}, 2	W_{out}, 2
MP2 Pooling size	(W_{out}, 2)	(W_{out}, 2)
DL Neurons	1	1

we assume that the sensors measurements of the SWaT and WADI datasets can be perturbed with Gaussian noise with mean $\mu = 0$ and standard deviation $\sigma \in \{1, 2, 3, 5, 10, 15\}$. The noise distribution and the values of μ and σ have been selected following similar assumptions as in other studies on noisy networked control systems [100, 101].

The validation presented below is divided into three different experiments. In Experiment 1 and 2, we compare the performance of DAICS with relevant works in the state-of-the-art in terms of precision, recall, F1 score, and the number of attacks correctly detected. Experiment 1 evaluates DAICS using the SWaT dataset, while experiment 2 uses the WADI dataset. In Experiment 3, we evaluate the robustness of DAICS to additive noise applied on the SWaT and WADI test sets. We also compare the results with the frameworks proposed in [14, 37] in case of the SWaT dataset.

3.4.3 Experiment 1: Detection accuracy in the SWaT

As shown in Table 3.6, DAICS outperforms existing state-of-the-art detection on the SWaT test set with 88.9% F1 score, and correctly recognising 33 out of 36 attacks in the test set. It is worth noting that DAICS also detects a higher rate of attacks during their execution (97% against 93%

of [37]), hence allowing the operator to activate the adequate countermeasures more promptly. A full comparison between DAICS, AADS [14] and other state-of-the-art solutions considered in this experiment is provided in Table 3.7.

Table 3.6: State-of-the-art comparison (SWaT dataset). In parenthesis, the number of attacks detected after their end

Architecture	**Precision**	**Recall**	**F1**	**Detected attacks**
DAICS	0.9185	0.8616	0.8892	33(1)
AADS [14]	0.866	0.861	0.863	33(1)
MLP [49]	0.967	0.696	0.812	25
CNN [37]	0.867	0.854	0.860	31(2)
TABOR [46]	0.861	0.788	0.823	24
Windowed-PCA [42]	0.92	0.841	0.879	-
Online-ADS [45]	0.8638	0.9064	0.8846	-

Missing attacks. The three attacks missed by DAICS (all attacks are targeting motorised valves) are marked as unsuccessful in the dataset documentation, as they do not cause any noticeable changes to the ICS. In attack #4, the attacker opens the motorised valve MV504 with the goal of halting the shut-down sequence of the Reverse Osmosis (RO) unit and reducing the life of RO unit. As denoted in the dataset documentation, this attack is unsuccessful, and it does not cause any notable changes in the ICS. Attack #13 was carried out by closing the motorised valve MV304; however, the dataset records analysis reveals that this valve was already closed before the start of the attack, causing the attack to produce no effects.

The attacker carried out attack #14 by keeping the motorised valve MV303 closed for 8 minutes when the valve was supposed to open. The goal of the attack was to halt stage 3 of the SWaT testbed; however and as mentioned in the dataset documentation, the attack failed because tank

Table 3.7: Recall for each attack in the SWaT dataset

Attack scenario[1]	**Attack description**	**MLP [49]**	**TABOR [46]**	**CNN [37]**	**AADS [14]**	**DAICS**
1	Open MV-101	0	0.049	0.995	0.953	0.879
2	Turn on P-102	0.764	0.930	1	1	1
3	Increase LIT-101 by 1mm every second	0	0	0.225[2]	0.296	0.075
4	Open MV-504	0	0.328	0	0	0
6	Set value of AIT-202 at 6	0.952	0.995	0.903	0.903	0.863
7	Water level LIT-301 increased above HH	0.909	0	1	0.012	0.032
8	Set value of DPIT as 40kpa	0.984	0.612	1	0.969	0.984
10	Set value of FIT-401 at 0.7	0.976	0.994	1	0.931	0.928
11	Set value of FIT-401 at 0	0.989	0.998	1	1	1
13	Close MV-304	0	0	0	0	0
14	Do not let MV-303 open	0	0	0	0.012	0
16	Decrease water level LIT-301 by 1mm each second	0.60	0	0.236	0	0.109
17	Do not let MV-303 open	0	0.597	0.631	0.539	0.577
19	Set value of AIT-504 at 16 uS/cm	0.97	0.004	0[2]	0.434	0.421
20	Set value of AIT-504 at 255 uS/cm	0	0.997	1	0.973	0.939
21	Keep MV-101 on continuously; set value of LIT-101 at 700mm	0.98	0.083	0.907	0.908	0.029
22	Stop UV-401; set value of AIT502 at 150; force P-501 to remain on	0.978	0.998	1	0.958	0.959
23	Set value of DPIT301 at 0.4 bar; keep MV302 open, P602 closed	0.711	0	1	0.954	0.984
24	Turn off P-203 and P-205	0.918	0	0.169	0.225	0.253
25	Set value of LIT-401 at 1000; P402 is kept on	0.294	0	0.019	0.899	0.479
26	P-101 is turned on continuously; set value of LIT-301 at 801mm	0.998	0.999	1	0.766	0.764
27	Keep P-302 on; set value of LIT401 at 600mm until 1:26:01	0	0.196	0.063	0.547	0.489
28	Close P-302	0.0324	0.936	1	1	1
29	Turn on P-201; turn on P-203; turn on P-205	0.87	0	0	0[2]	0[2]
30	Turn P-101 and MV-101 on; set value of LIT-101 at 700mm;	0.834	0.999	1	1	1
31	Set LIT-401 at less than L	0.786	0	0.303	0.626	0.874
32	Set LIT-301 at above HH	-	0	0.935	0.838	0.539
33	Set LIT-101 at above H	-	0.890	0.883	0.151	0.16
34	Turn P-101 off	0.331	0.990	0.6	0.01	0.52
35	Turn P-101 off; keep P-102 off	0.84	0.258	0	0.042	0.041
36	Set LIT-101 at less than LL	0.808	0.889	0.878	0.9	0.842
37	Close P-501; set value of FIT-502 at 1.29 at 11:18:36	0.842	0.998	0.895	0.915	0.847
38	Set value of AIT402 at 260; set value of AIT502 at 260	0.767	0.996	0.864	1	0.977
39	Set value of FIT-401 at 0.5; set value of AIT-502 at 140 mV	0.836	0.369	0.908	1	0.944
40	Set value of FIT-401 at 0	0.784	0.997	1	1	1
41	Decrease LIT-301 value by 0.5mm per second	0	0	0.638	0.626	0.925

[1] Identifiers of attack scenarios from the swat dataset documentation.
[2] The attack is detected after its end.

T3 was already full. Figure 4.5 confirms the stable water level in the tank during the attack. Furthermore, as illustrated in Figure 4.6, the prediction error is well below the threshold.

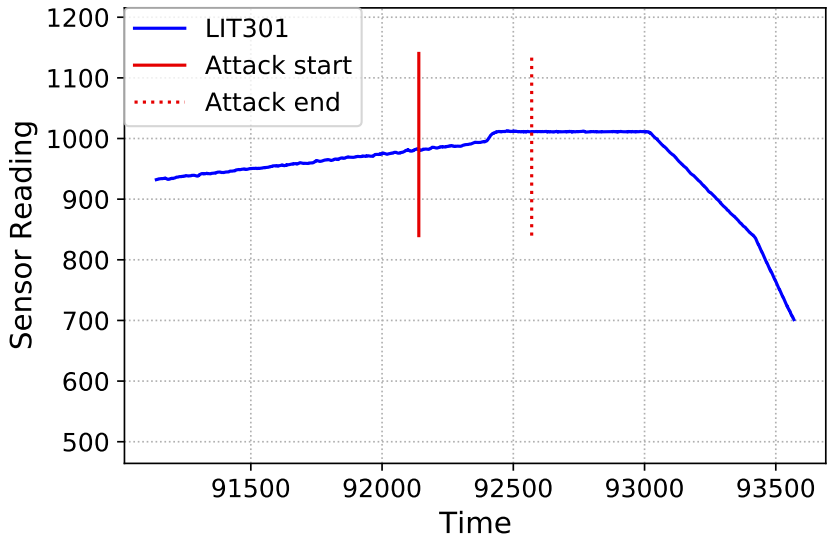

Figure 3.7: The reading of the water level sensor LIT301 before and after attack scenario #14.

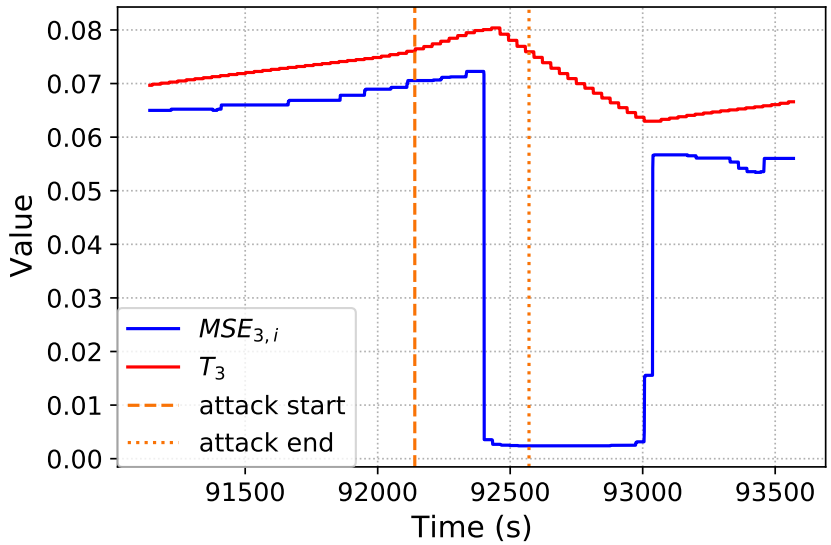

Figure 3.8: The prediction error during attack scenario #14. As can be seen, the prediction error does not exceed the threshold.

Few-time-steps algorithm. We also evaluated the contribution of the

few-time-steps algorithm to the detection accuracy of DAICS. To this aim, we repeated the experiment disabling the algorithm. As the SWaT test set contains normal records that are not present in the training set, the model of normal behaviour built with the training set leads to several false positives and, more precisely, to a low F1-score measure of 77.7% and 1251 false alarms (compared to the 645 experienced with the algorithm enabled).

The update process is fast. Indeed, with our setup, the execution of one cycle of the few-time-steps learning algorithm, including the $EP_{tuning} = 100$ epochs, with a batch of 32 samples takes 400 ms on average. In an online system, the algorithm can be triggered by the technician upon identifying a false alarm, e.g. caused by a known unstable device. It is important to emphasise that this is the only requirement for the technician, unlike other approaches in which an in-depth knowledge of the underlying algorithms is necessary to update detection thresholds or other parameters.

Hyper-parameters. The sensitivity of DAICS to the hyper-parameter W_{anom} is presented in Figure 3.9. We remind that W_{anom} is defined in Equation 3.3 as the number of consecutive times the prediction error MSE is higher than the anomaly threshold on the same output section of WDNN.

Although the number of detected attacks is quite stable to 33 (except for W_{anom}=36), the highest F1 score (0.8892) is measured at W_{anom}=30.

False alarms. DAICS relies on the feedback from the technician to fine-tune the output sections with new information of the normal behaviour. Of course, the rate of technician's interventions is a relevant metric to consider, as too frequent false positives can undermine the usability of the system. With the chosen settings, we measured 645 false alarms in total on the test set of the SWaT dataset, equivalent to 6.6 human interventions/hour on average. Although this seems a manageable rate of intervention, in more complex industrial environments compared to the SWaT testbed, this rate should be reduced.

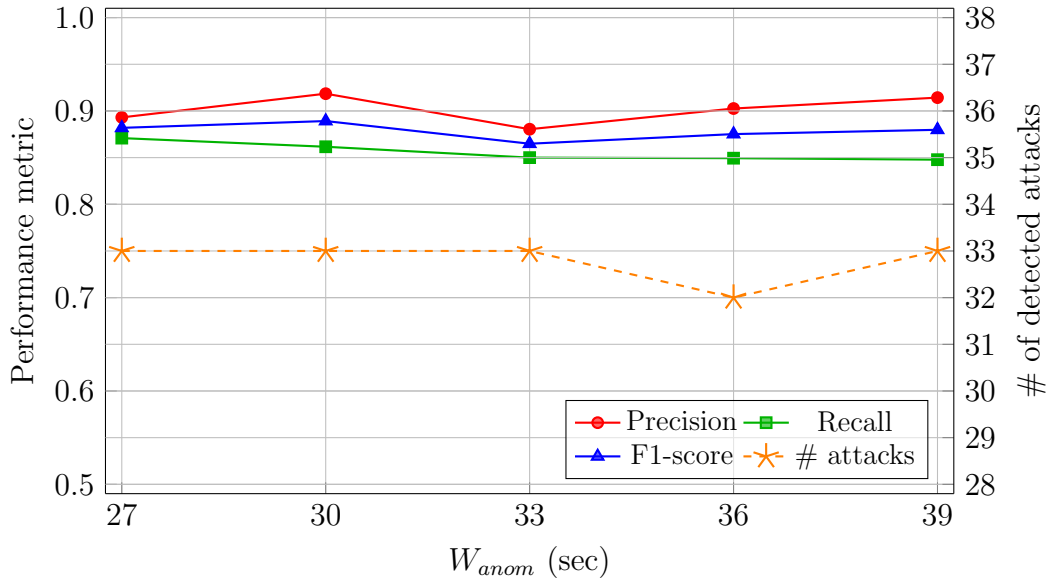

Figure 3.9: Sensitivity of WDNN to W_{anom} on SWaT dataset.

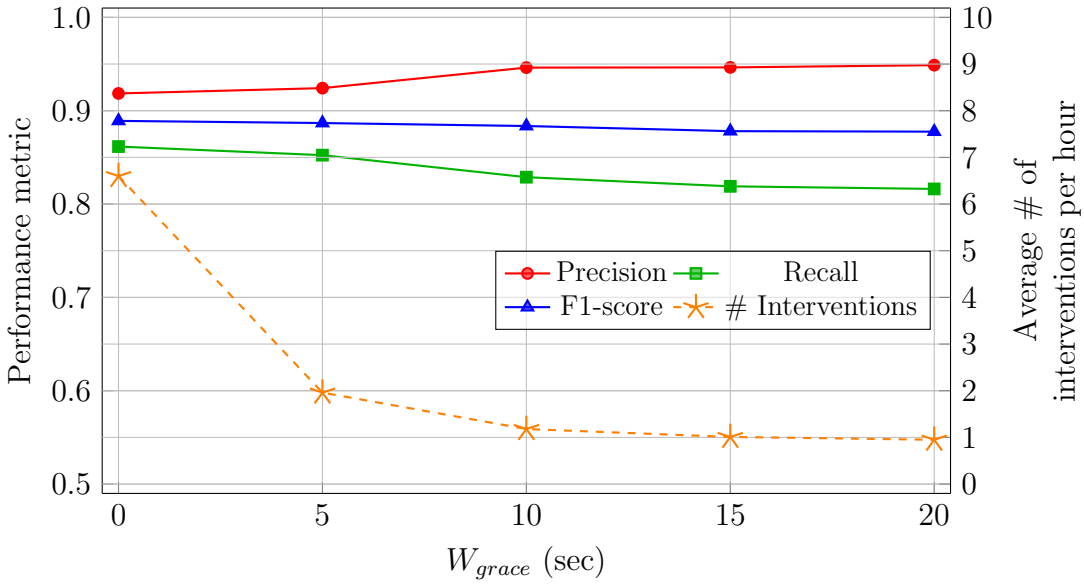

Figure 3.10: Sensitivity of WDNN to W_{grace} on SWaT dataset.

One way to reduce the rate of intervention is to allow a *grace time* W_{grace}, through which alarms are reported to the technician only if they last for at least W_{grace} seconds. Of course, this practice may hide short true positives, hence preventing the detection of a certain number of anomalies.

We tested the sensitivity of DAICS to the grace time by varying W_{grace}

from 1 to 20 seconds. In Figure 3.10 we can observe a minimal decrease of the F1 score (0.877 with W_{grace}=20) and a consistent reduction of the technician intervention rates (from 6.6/hour with no grace time, to 1.96/hour with W_{grace}=5, to 0.95/hour with W_{grace}=20). Moreover, we did not experience any variations in the number of detected anomalies with W_{grace}=5 (still 33), while with W_{grace}=20 this number decreases to 28, as somehow expected.

3.4.4 Experiment 2: Detection accuracy in the WADI

DAICS matches the state of the art results in terms of number of attacks detected, 14 out of 15 attacks in the WADI dataset, while it outperforms the other solutions in terms of precision, recall and F1 score, as summarised in Table 3.8.

Missing attack. DAICS does not detect attack #8. In this attack, the attacker opens the motorised valve 2MCV007 to produce water leakage before the water reaches the consumer tanks. However, this attack is described as unsuccessful in the dataset documentation, meaning that it does not produce any effects on the ICS's operations.

Table 3.8: State-of-the-art comparison (WADI dataset)

Architecture	Precision	Recall	F1	Detected attacks
DAICS	0.9083	0.7205	0.8036	14
AADS [14]	0.7794	0.6591	0.7142	14
MAD-GAN [102]	0.4144	0.3392	0.37	-
1D CNN [42]	0.697	0.731	0.714	14
AE [42]	0.834	0.681	0.750	14

Few-time-steps algorithm. Given the setup described in section 3.4.1, DAICS takes 800 ms to fine-tune an output section with 100 epochs and a batch of 32 samples. Like in the case of the SWaT dataset, we demonstrate how the few-time-steps algorithm contributes to keeping weights and biases

of WDNN up-to-date with the changes of the normal behaviour. Indeed, when disabling the updating mechanism, the F1-score drops to 0.5621, while the number of false alarms increases from 226 to 684.

Hyper-parameters. Increasing the anomaly window W_{anom} determines an increase on the number of detected attacks, as shown in Figure 3.11. This is mainly due to the condition $\text{MSE}_{g,i} > T_g$ in Equation 3.3, which yields to the detection of short attacks when W_{anom} is large enough. However, for the state-of-the-art comparison reported in Table 3.8, we set W_{anom}=30, as this value maximises the F1 score.

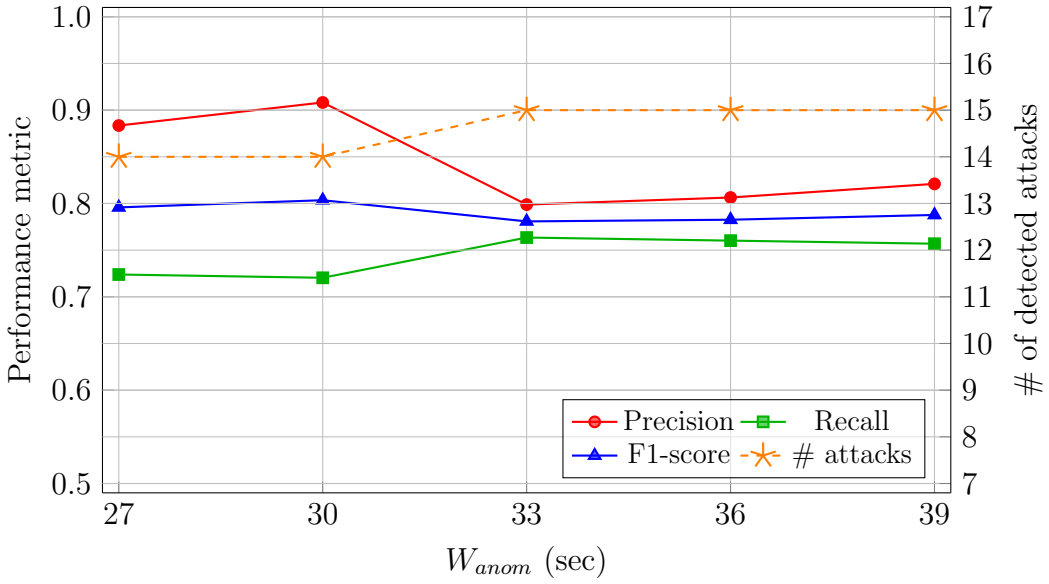

Figure 3.11: Sensitivity of WDNN to W_{anom} on WADI dataset.

False alarms. With the settings listed in Table 3.3, DAICS produces 226 false positives on the WADI test set, equivalent to 5.3 human interventions per hour on average. As for the SWaT dataset, we measured the sensitivity of DAICS to the *grace time* W_{grace}. The results, reported in Figure 3.12, show that increasing W_{grace} reduces the number of interventions to 1.95 per hour on average at W_{grace}=20, although negatively impacting on precision, recall and F1 score even at W_{grace}=5.

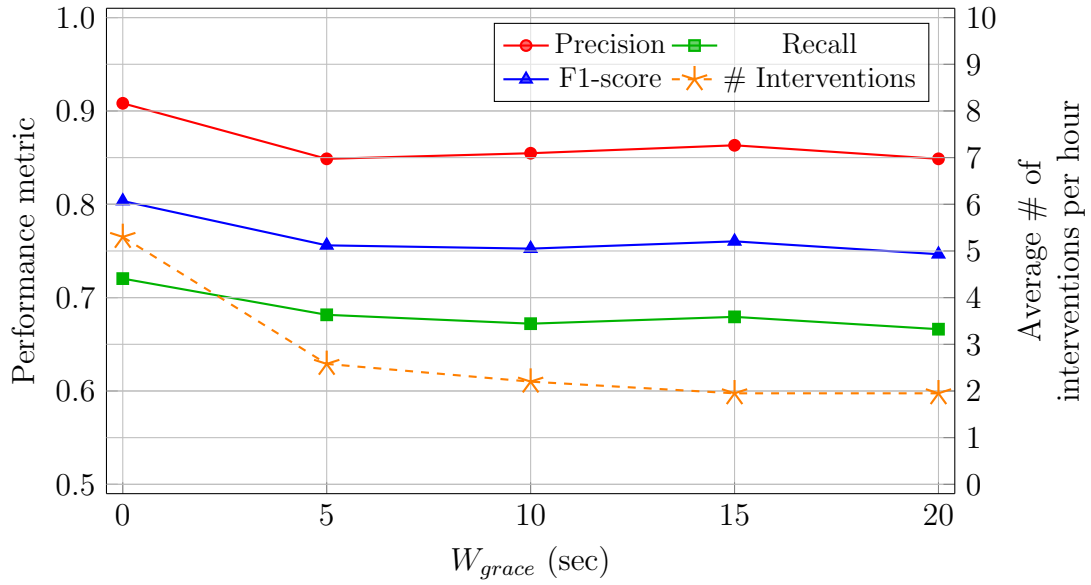

Figure 3.12: Sensitivity of WDNN to W_{grace} on WADI dataset.

3.4.5 Experiment 3: robustness to additive noise

In this experiment, we measure the robustness of DAICS to noise added to the sensors readings. In our experiments, we add various levels of synthetic noise to the sensor readings of both SWaT and WADI datasets. As anticipated in Section 3.4.2, we use white Gaussian noise with mean $\mu = 0$ and increasing standard deviation $\sigma \in \{1, 2, 3, 5, 10, 15\}$. We then observe the behaviour of DAICS with respect to the number of detected attacks and the detection accuracy measured with the F1 score metric. We then extend the experiment 2 in [14], where AADS and the CNN proposed in [37] are compared, by adding the performance of DAICS on the SWaT dataset. Note that, we have implemented the best CNN architecture as detailed in the original paper [37].

Figure 3.13 reports on the performance of DAICS as a function of the Gaussian noise level. Although the trend of the point-based F1 score is similar on both datasets, we can observe a higher impact of the synthetic noise on the detection of attacks in the SWaT dataset. We recall that an

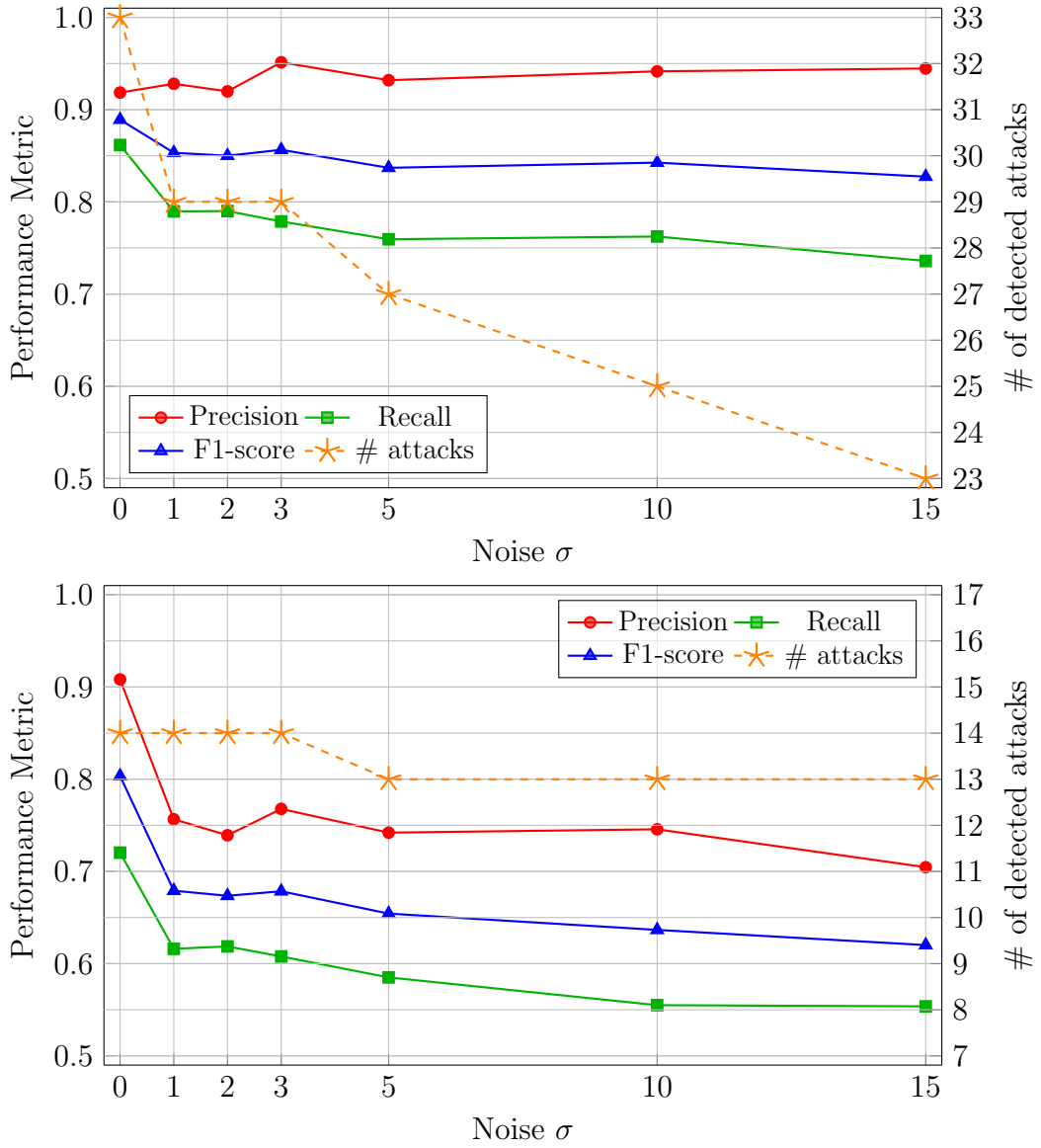

Figure 3.13: Performance of DAICS when adding Gaussian noise to the SWaT (above) and WADI (below) test sets.

attack is determined when the conditions in Equation 3.3 are met. In the case of sensors, the MSE must be above the threshold for W_{anom} consecutive samples. However, the increasing number of point-based false negatives introduced with the noise, as shown in the two figures by the trend of the recall measure, reduces the chances of finding W_{anom} consecutive point-based positive samples. Of course, this affects more the detection of short

attacks, which are more frequent in the SWaT dataset.

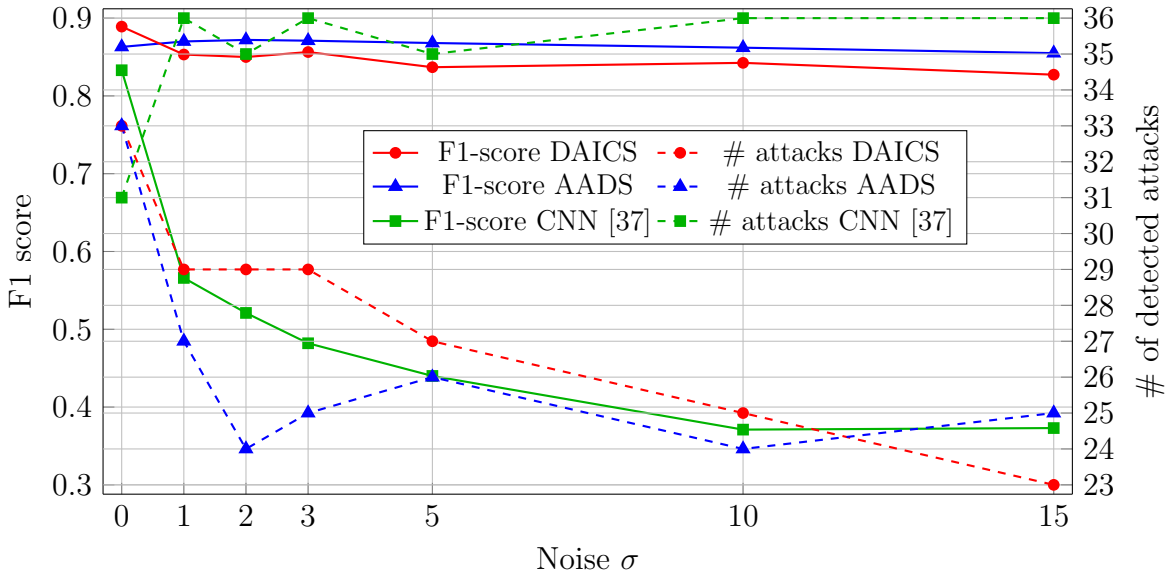

Figure 3.14: Performance when adding Gaussian noise to the SWaT test set.

Figure 3.14 shows the values of F1-score and number of detected attacks as functions of σ. We can notice that DAICS improves AADS in the number of detected attacks at any level of noise, except for σ=15 while almost matching the F1 score (we measured a slight decrease of 0.015 on average for $\sigma > 0$). This demonstrates that DAICS is also solid in noisy conditions. On the contrary, the F1 score of the state of the art CNN drops drastically (green solid curve). This is mainly due to the statistical approach and to the static threshold employed to detect the anomalies, empirically selected as the value $T \in [1.8, 3]$ that maximises the F1-score. The low values of the F1-Score for $\sigma > 0$ are due to a low precision score (around 0.31 on average), meaning that the CNN classifies most of the records as anomalies, including normal samples. This is the reason why the 36 attacks in the SWaT test set are almost all correctly classified (green dashed line). However, as everything looks like an anomaly, in a real-world deployment the output of this approach would be unusable.

In conclusion, DAICS outperforms state-of-the-art solutions on both

SWaT and WADI datasets with respect to point-based F1 score and number of detected attacks. In addition, DAICS is more robust to the additive noise, hence potentially more reliable in real-world industrial scenarios where the electromagnetic noise and the natural degradation of the devices might result in noisy data.

3.5 Summary

In this chapter, DAICS, a framework for anomaly detection in ICSs based on a one-class classification paradigm and unsupervised learning, was presented. The framework was designed to increase the robustness to non-adversarial input perturbations by addressing three major limitations of state-of-the-art solutions. The first limitation is vulnerability to perturbations caused by normal behaviour evolution, which is observed when the normal behaviour of the industrial process evolves over time. If not handled correctly, this phenomenon may be a source of false alarms. The second limitation is the vulnerability to noisy data, caused either by interference in the communication channel within the ICS or by ageing devices, which prevents the detection system from correctly differentiating anomalies from normal operations. The last limitation is the static thresholds that are manually fine-tuned at the test time, which may require long tuning sessions and are impractical in a live industrial process.

To tackle the aforementioned problems, in this study, a fast and automated mechanism was introduced to update the ML model based on the detected false alarms caused by changes in normal behaviour. Such a mechanism, which is based on the so-called *few-time-steps* algorithm, has been designed for use in production environments, with the only assumption being that technicians can identify false alarms caused by maintenance operations (hardware/software updates) or known misbehaving devices and

signal them to DAICS. This sets DAICS apart from similar solutions, as the other solutions require in-depth knowledge of the underlying algorithms to update thresholds or other parameters. The combination of the few-time-steps algorithm with a dynamic threshold mechanism, also proposed in this chapter, allows DAICS to outperform state-of-the-art solutions in terms of the number of detected attacks and resilience to noisy data samples.

Chapter 4

Similarity-based false alarm reduction for ADSs

In the previous chapter, it was proven that anomaly detection systems that are robust to non adversarial input perturbations can be developed. However, the problem of human intervention required to verify false alarms remains unaddressed. Indeed, the DAICS framework requires at least two interventions per hour in a small scale testbed, similar to the SWaT. In a city-scale water treatment plant, such a false alarm rate might overwhelm the system operator and reduces the trust in the anomaly detection solution. These false alarms can be triggered by a gradual concept drift in the normal behaviour of the ICS. Gradual drift usually occurs because of moderate environmental and operational changes [103]. Previous studies aiming to reduce false alarms were either based on complicated neural networks, e.g., Bidirectional Encoder Representations from Transformers (BERT), or unsuitable for ICSs [103–106].

In this chapter, we propose an extension of DAICS, called Similarity based technique for reduction of False Alarms (SiFA). SiFA collects a buffer of historical false alarms and suppresses every new alarm that is similar to these false alarms. The buffer is initialised with data samples that cause false alarms in a validation dataset, which comprises only benign

data samples. This buffer is then updated (a new instance is added or an old instance is replaced) whenever the operator identifies a new false alarm. When an alarm condition is detected using the DAICS anomaly detection logic, the similarity between this alarm and every instance in the buffer is computed using a distance function, such as the Euclidean distance or cosine distance. The minimum computed distance is compared to a similarity threshold, above which the new alarm is considered a true alarm and is thus reported to the operator. Below this threshold, the alarm is considered similar to at least one historical false alarm, and is thus suppressed. SiFA is evaluated using a dataset collected from the SWaT testbed. The dataset comprises a training set with only normal records and a test set with normal and anomalous records. The anomalies in the test set are real-world attacks targeting the integrity and availability of the testbed. A portion of the training set is used as a validation set to tune the hyperparameters and initialise the buffer of the false alarms.

The rest of this chapter is organised as follows: Section 4.1 presents the problem statement. The SiFA approach is explained in Section 4.2. The experimental setup, methodology, and results are provided in Section 4.3. Finally, Section 4.4 summarises this chapter.

4.1 Problem statement

A major challenge to ADSs is the non-adversarial input perturbations that cause the normal behaviour of the monitored system to evolve over time, generating false positives as novel behaviours are considered anomalous. These false positives require human intervention, which is a burden on the system operator and may render the ADS unusable. In Chapter 3, the DAICS framework is presented to address the problem of non-adversarial input perturbations. DAICS reduced the number of false positive alarms

and, therefore, the number of human interventions to approximately 6.6 alarms/hour. This number of false alarms was further reduced by introducing the concept of grace time W_{grace}, after which the alarm is reported to the operator. In other words, an alarm is reported only if it lasts for at least W_{grace} seconds.

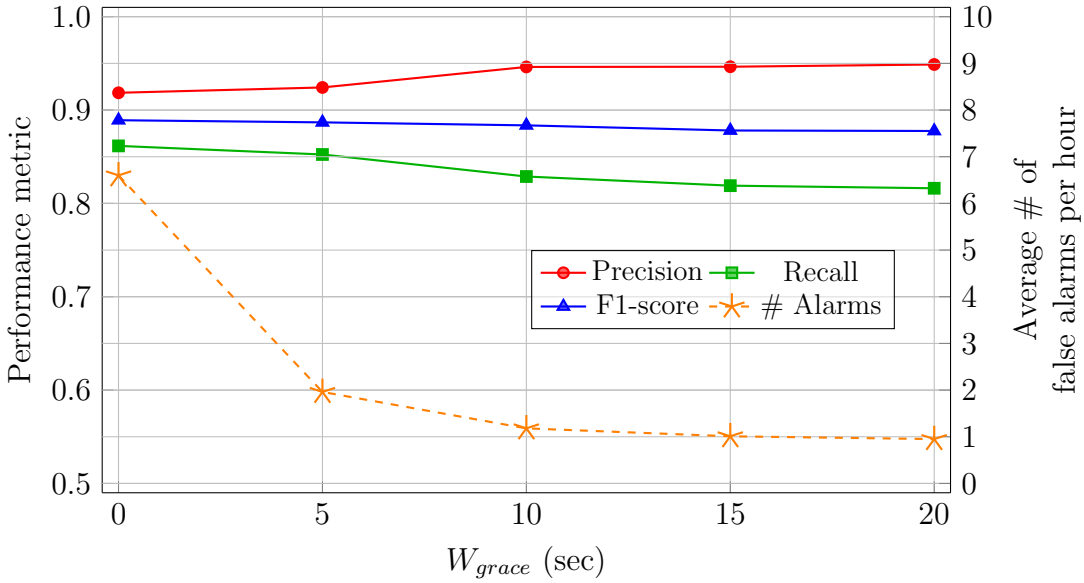

Figure 4.1: Sensitivity of DAICS to W_{grace} on SWaT dataset.

Figure 4.1 shows the sensitivity of DAICS with respect to W_{grace} for the SWaT dataset. As can be observed, the number of false alarms is reduced to 1.96 alarms/h without reducing the detection accuracy with $W_{grace} = 5s$. However, 1.96 alarm/h can be considered a high rate, given the small scale of the SWaT testbed. Furthermore, the detection accuracy is impacted when $W_{grace} > 5s$ because attacks with short durations are not signalled. As a result, decreasing the number of false-positive alarms without affecting detection accuracy remains a challenge for the implementation of the anomaly detection solutions in the real-world.

The next section presents SiFA, which is an extension of DAICS. SiFA is a solution for detecting false alarms before they are reported to the operator, and thus reduces the number of human interventions required.

The proposed approach collects a buffer for historical false alarms and suppresses every new alarm similar to these false alarms.

4.2 The SiFA framework

As mentioned earlier, SiFA collects a buffer of sensor readings that caused false alarms in the past and suppresses every new alarm that is highly similar to these false alarms, eliminating the burden of analysing such alarms on the technician. SiFA is an extension of the DAICS framework [15], and, thus, the anomaly detection logic, neural network architecture and anomaly threshold tuning logic are the same as those in DAICS. The difference is that the *few-time-steps* algorithm in DAICS is replaced by the SiFA false alarm detection algorithm. To help the reader to recall the some necessary details, an introduction to the neural network and the actuator database is provided in the following subsection.

4.2.1 Neural network architecture

The neural network takes as input an array X of data samples collected during a time window of length W_{in} seconds, corresponding to W_{in} samples, as the dataset was collected at a sampling rate of one sample per second. The size of X is $m \times W_{in}$, where $m = m_{se} + m_{ac}$ is the number of features for each sample including the state of m_{se} sensors and m_{ac} actuators taken at time t. Observe that we use the states of both sensors and actuators because the future states of sensors depends on their current states and the actions taken by actuators. The output Y is the expected normal states of sensors during a future time window W_{out}. Both time-windows W_{in} and W_{out} are separated by a time interval called horizon H. The purpose of H is to prevent the neural network from replicating the last values of the input time window W_{in} into W_{out}, as pointed out by Shalyga et al. [49].

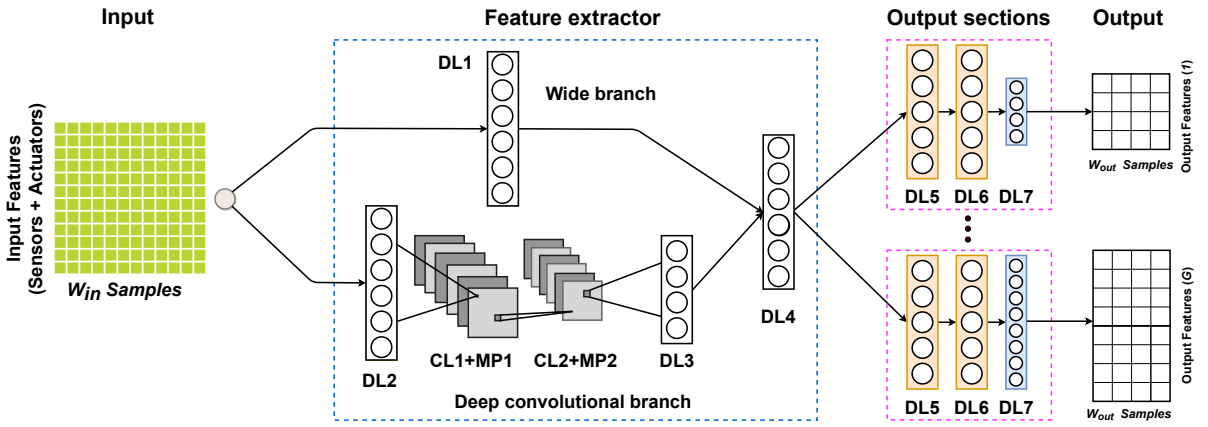

Figure 4.2: Architecture of WDNN. The prediction of the sensors states is computed combining the states of both sensors and actuators. The size of DL7 varies according to the number of output features of the output section.

As shown in Figure 4.2, the *Feature extractor* section comprises two convolutional layers and four fully connected layers organised into two branches. The fully connected layer DL1 represents the *wide branch*, which learns the normal interactions between sensors and actuators using cross-product transformations between the input features.

The *deep branch* allows the WDNN to generalise to events not covered in the training set, hence to reduce the prediction error in the case of new normal combinations of input states. This branch comprises the fully connected layer (DL2), two one-dimensional convolutional layers (CL1 and CL2), each one followed by a max-pooling layer (MP1 and MP2), and the fully connected layer (DL3). As shown in other works (e.g., [37]), one-dimensional CNNs are particularly suited for modelling time series data. The purpose of layers CL1 and CL2 is to model the data collected from sensors and actuators in a specific time window. With max-pooling, we down-sample the output of each convolutional layer by a factor of 2. Finally, the fully connected layer DL3 re-shapes the output of MP2 to allow its concatenation with the output of the wide branch.

Both branches are aggregated in another fully connected layer (DL4)

followed by multiple dense output sections, each one consisting of the three fully connected layers DL5, DL6 and DL7. Each output section learns the most relevant information from the aggregation layer in order to predict the normal behaviour of a group of sensors controlled by a specific PLC.

4.2.2 Anomaly detection in actuators

The SWaT testbed include discrete actuators such as pumps and motorised valves. The pumps are arranged in pairs of primary and redundant hot-standby pumps. A redundant pump is turned on only in the case the respective primary pump stops working. This operational mode complicates building a forecasting model that predicts the actuator states, as some actuators (such as the redundant pumps), are rarely used during the normal operations. As a consequence, after measuring a high prediction error with DL-based methods due to lack of normal operation records, we designed a light and straightforward approach based on querying a database containing all the normal actuator states. A database entry is n-tuple, where n is the number of actuators in the testbed ($n = 26$ in the SWaT). Each entry is a combination of actuators states labelled as normal, for a total of 146 entries available in the SWaT dataset. An example of tuple from the SWaT testbed is provided in Table 4.1.

Table 4.1: Tuple in the database of actuators normal states

Actuator	MV101	P101	P102	...	P601	P602	P603
Tuple	2	2	1	...	1	1	1

At testing time, the combinations in the test set with no occurrences in the training set are marked as anomalies.

Algorithm 3 Create alarm instance algorithm

1: **procedure** CREATEINSTANCE(Sensors readings of section g (Y^g), Buffer maximum length (L_{buffer}))
2: $x.sensors = Y^g$
3: $x.rscore = L_{buffer}$
4: $x.fscore = 0$
5: **return** x
6: **end procedure**

4.2.3 The alarm class

SiFA collects a buffer of false alarms B^g and then suppresses every new alarm similar to these false alarms. Thus, B^g is basically a list of alarm instances. Moreover, an alarm instance is created upon the detection of a new alarm condition using the DAICS anomaly detection logic. This alarm instance is matched with every instance in the buffer and suppressed if a match is found. Algorithm 3 describes the steps involved in creating an alarm instance. Specifically, alarm instance x comprises three attributes: $x.sensors$, $x.rscore$ and $x.fscore$. $x.sensors$ is the sensor readings Y^g of output section g that caused the alarm and was collected during a time window W_{out}. Therefore, $x.sensors$ is a $m^g_{se} \times W_{out}$ array, where m^g_{se} is the number of sensors in the output section g. $x.rscore$ is a recency score representing how recently an alarm was added to the buffer. $x.fscore$ is a frequency score that represents how often this false alarm is matched with new alarms. The default value of the recency score $x.rscore$ is the maximum buffer length L_{buffer} which indicates that this instance is the most recent in the buffer. On the other hand, the default value of the frequency score $x.fscore$ is zero, indicating that this false alarm did not match with any new alarms.

Algorithm 4 Calculate RF scores algorithm

1: **procedure** RFScores(Buffer of section g (B^g))
2: $scores = []$
3: **for** $i = 0$ **to** $len(B^g)$ **do**
4: $rf = int(concatenate(B^g[i].rscore, B^g[i].fscore))$
5: $scores = scores \cup rf$
6: **end for**
7: **return** *scores*
8: **end procedure**

4.2.4 Recency frequency scores

The recency and frequency scores are used to calculate the so-called Recency Frequency (RF) score, which indicates how recent the alarm is and how frequently it matches new alarms. In Algorithm 4, both scores are concatenated into a single-integer value. The algorithm accepts as input a buffer of a given output section B^g and returns a list of scores that indicate the importance of every alarm in the buffer. An instance with high recency and frequency scores is assigned a high RF score, whereas an instance that had not matched any new alarms recently is assigned a low RF score.

4.2.5 Similarity threshold

Algorithm 5 Calculate Similarity Threshold algorithm

1: **procedure** SimThreshold(Buffer of section g (B^g))
2: $dist \leftarrow$ Pairwise squared Euclidean distance between $x.sensors$ in B^g
3: $min\_dist \leftarrow$ Use $dist$ to find the minimum pairwise distances ▷ Array of L_{buffer} items
4: $\lambda^g = Mean(min\_dist)$
5: **return** λ^g
6: **end procedure**

When the anomaly detection logic detects a new alarm, an alarm instance is created and the $x.sensors$ attribute is compared with every instance in buffer B^g. The comparison between every two $x.sensors$ attributes is

performed using a similarity metric that is a distance function. Therefore, a threshold at such a distance is needed, such that the two compared alarms are identical if the distance is less than or equal to the threshold. The threshold λ^g is calculated using Algorithm 5. In line 2, the pairwise squared Euclidean distance is computed between every $x.sensors$ pair in the buffer, and the output is a square matrix with dimension L_{buffer} and stored in the *dist* variable. The squared Euclidean distance function was selected empirically in this study; however, other similarity functions can be used. In the *dist* matrix, there is a minimum distance associated with each instance. These minimum distances can be calculated across the rows or columns of the matrix (line 3). In line 4, the average of these minimum distances is computed and used as the similarity threshold λ^g.

4.2.6 Buffer initialisation and update

The false alarm buffer is initialised with false alarms generated on a validation dataset. Subsequently, the buffer is updated when the technician detects a new false alarm. As shown in Algorithm 6, an alarm instance is directly added to the buffer if it is empty (lines 2-3). If the buffer is not empty and the number of instances is less than the maximum buffer length, a new false alarm is added to the buffer (lines 4-12). Specifically, in line 5, the algorithm verifies that the sensor readings $x.sensors$ do not already exist in the buffer. In lines 6-7, the recency scores of every instance in the buffer are decremented to indicate that they are older than the new instance. Subsequently, the newly created instance x is added to the buffer. Note that the recency score of this instance is the default value L_{buffer}, indicating that it is the most recent instance in the buffer. In line 10, the similarity threshold λ^g is updated to reflect addition of a new false alarm.

If a new false alarm is detected by the technician (or in the validation dataset) and the buffer is full, the alarm replaces the buffer instance with

Algorithm 6 Buffer-init-update algorithm

Input: False alarm instance x in output section g, Buffer of section g (B^g), Buffer maximum length (L_{buffer})
Output: Updated buffer (B^g), Updated similarity threshold (λ^g)

```
1: # A new false alarm is detected on the validation dataset or by the technician
2: if len(B^g) == 0 then
3:     B^g = [x]
4: else if len(B^g) < L_buffer then
5:     if x.sensors ∉ B^g then
6:         for i = 0 to len(B^g) do
7:             B^g[i].rscore = max(B^g[i].rscore − 1, 0)
8:         end for
9:         B^g = B^g ∪ x
10:        λ^g = SimThreshold(B^g)
11:    end if
12: else
13:    if x.sensors ∉ B^g then
14:        scores = RFScores(B^g)
15:        idx_min = arg_min(scores)          ▷ Find index of the minimum score
16:        for i = 0 to len(B^g) do
17:            B^g[i].rscore = max(B^g[i].rscore − 1, 0)
18:        end for
19:        B^g[idx_min] = x
20:        λ^g = SimThreshold(B^g)
21:    end if
22: end if
```

the lowest RF score (lines 12-24). Specifically, in line 13, the algorithm verifies that sensor readings of the new alarm do not already exist in the buffer. The RF score of every instance in the buffer is computed, and the index of the lowest score is found (lines 14-15). Then, the recency score for every instance in the buffer is decremented (lines 16-18). Finally, in lines 19-20, the newly created instance replaces the instance with the smallest RF score in the buffer and the similarity threshold is updated based on the new instance.

Algorithm 7 False-Alarm-Detection algorithm

Input: Batch of samples (S), Buffer (B^g) $g \in [1, G]$, Similarity threshold (λ^g) $g \in [1, G]$, Alarm at time t ($A_t[g]$) $g \in [1, G]$, Buffer maximum length (L_{buffer})
Output: Retrained output section ($\mu_{out}[g]$), Updated buffer B^g

1: **for** $g \in [0, G]$ **do**
2: $Y^g \leftarrow$ Sensor readings of section g from batch S
3: $x =$ CreateInstance(Y^g, L_{buffer})
4: $dist\_euc \leftarrow$ Squared Euclidean distance between $x.sensors$ and every instance in B^g
5: $idx\_min \leftarrow$ Index of the minimum distance in $dist\_euc$
6: **if** $dist\_euc[idx\_min] <= \lambda^g$ **and** $\nu_t \in A$ **then**
7: # The alarm $A_t[g]$ is similar to a historical false alarm and will be suppressed
8: **for** $i = 0$ **to** $len(B^g)$ **do**
9: **if** $i == idx\_min$ **then**
10: $B^g[i].rscore = min(B^g[i].rscore + 1, L_{buffer})$
11: $B^g[i].fscore = min(B^g[i].fscore + 1, L_{buffer})$
12: **else**
13: $B^g[i].rscore = max(B^g[i].rscore - 1, 0)$
14: **end if**
15: **end for**
16: $EP_{tuning} \leftarrow$ number of fine-tuning epochs;
17: $\mu_{out}[g] \leftarrow$ weights and biases
18: **for** epoch **in** EP_{tuning} **do**
19: $SGD(\mu_{out}[g])$ step to minimise the cost of section g on S;
20: $\mu_{out}[g] \leftarrow$ update weights and biases;
21: **end for**
22: **else** ▷ The alarm is not similar to historical false alarms
23: Report the alarm $A_t[g]$ to the technician
24: **if** $A_t[g]$ **is** False **then**
25: Call the buffer-init-update algorithm (Algorithm 6)
26: **if** $\nu_t \notin A$ **then**
27: $A = A \cup \nu_t$; ▷ Update the database of actuators
28: **end if**
29: Repeat steps 16-21
30: **end if**
31: **end if**
32: **end for**

4.2.7 Alarm matching

SiFA attempts to identify and suppress alarms that are similar to buffer B^g, and hence, reduces the number of alarms reported to the technician. Upon detection of an alarm condition using the DAICS anomaly detection logic, an alarm instance x is created and compared to the buffer of false alarms

using a distance (similarity) function. This distance function computes a similarity metric between the new alarm and each instance in the buffer. If the similarity metric is less than or equal to the threshold λ^g, the alarm is suppressed; otherwise, the alarm is reported to the operator.

Algorithm 7 describes aforementioned the process. In line 2, the sensors readings Y^g are extracted from the batch of samples that caused an alarm. Then, an alarm instance x is created, and the squared Euclidean distance between $x.sensors$ and every instance in the buffer is computed and stored (lines 3-4). The squared Euclidean distance is selected empirically in this study; however, other similarity functions (e.g., cosine similarity) can be used. In line 5, the index of the instance most similar to x (i.e., with the smallest distance) is found. If the smallest distance ($dist\_euc[idx\_min]$) is less than or equal to the similarity threshold λ^g and the actuator combinations belong to database A, the alarm is considered to match a historical false alarm and is thus suppressed (lines 6-21). In lines 8-15 of Algorithm 7, the recency and frequency scores of the buffer instance that matched the suppressed alarm are incremented, while the recency score of every other buffer instance is decremented. Therefore, an instance that is frequently matched has a high RF score and remains in the buffer. Recall that, Algorithm 6 replaces instances with a low RF score. In lines 16-21, the output section is tuned based on a batch of samples S that produces the alarm to learn from the ongoing gradual concept drift. Specifically, in lines 17-21, SGD is employed to fine-tune the output section through multiple gradient steps. We calculate the prediction loss for the data samples aggregated in a *batch* of S samples containing the false alarm (S is passed as an input to the algorithm). The optimiser minimises this loss by tuning the parameters of the output layers DL, DL6, and DL7. After EP_{tuning} optimisation steps (approximately 20 ms on average for $five$ epochs on our testing environment described in Section 4.3), the updated output section $\mu_{out}[g]$ replaces the

previous one in the anomaly detection process (line 20).

In lines 22-31, the smallest distance $dist\_euc[idx\_min]$ is greater than the similarity threshold λ^g or the actuator combinations do not belong to database A, and thus, the alarm is reported to the technician. If the technician decides that the alarm is false, Algorithm 6 is used to add a new alarm to the buffer. Furthermore, the actuator database A is updated with any new actuator combination ν_t and the output section g is tuned.

4.3 Experimental evaluation

In this section, a detailed evaluation of SiFA on the SWaT dataset is presented. The evaluation includes a comparison with DAICS in terms of the detection accuracy and number of detected attacks. Finally, a detailed analysis of the missing attacks is provided.

4.3.1 Experimental setup

SiFA was implemented in PyTorch and validated in a Google Colab notebook accelerated by a GPU. Given that the SiFA is an extension of DAICS, the rest of the implementation details are the same as in DAICS.

4.3.2 Methodology

The classification accuracy is evaluated using the F1 score, which combines both the recall and the precision and hence provides an overall measure of the model's performance. The performance improvement due to the detection (suppression) of false alarms can be indicated by the false positive rate (FPR), which is the ratio of misclassified normal samples to the total number of normal samples in the dataset. Furthermore, the false negative rate (FNR), which is the ratio of misclassified attack samples to the total

number of attack samples in the dataset, is also used to indicate whether the SiFA is missing attack samples. These metrics are formally defined as follows:

$$Pr = \frac{TP}{TP + FP} \quad Re = \frac{TP}{TP + FN}$$
$$FNR = \frac{FN}{FN + TP} \quad FPR = \frac{FP}{FP + TN} \quad F1 = 2 \cdot \frac{Pr \cdot Re}{Pr + Re}$$

These metrics are point-based and are thus computed using the labelled samples (points) of the dataset, where each sample is a combination of sensors and actuators collected at a given time.

The hyperparameters of DAICS are the same as in [15] except for the number of tuning epochs ($EP_{tuning} = 5$). SiFA has one hyperparameter, that is, the maximum buffer length $L_{buffer} \in \{100, 150, 177, 200, 250\}$, where 177 is the number of false alarms collected on the SWaT validation dataset and used as the default value of L_{buffer}. The other values of L_{buffer} are used to analyse the sensitivity of the SiFA to this hyperparameter.

4.3.3 Experimental results

DAICS reduced the number of human interventions due to false alarms by introducing the concept of grace time, after which an alarm is reported to the system operator, i.e., an alarm is reported if it lasts for at least W_{grace} seconds. The evaluation results of DAICS revealed that false alarms drop from 6.6/h with no grace time to 1.96/h with $W_{grace} = 5s$. However, this practice may hide the alarms for attacks of short duration. Therefore, the SiFA approach was presented as an extension of the DAICS framework to avoid ignoring alarms based on their duration. SiFA relies on measuring the similarity between every new alarm and a buffer of historical false alarms. The buffer is initialised with false alarms generated on the validation SWaT dataset, which contains only normal samples. The count of these false

alarms is 177, and they are used to populate the buffers of all output sections.

As shown in Table 4.2, at a buffer length $L_{buffer} = 177$, which is the default value, SiFA improves the performance of DAICS with an F1 score $> 91\%$. Furthermore, false alarms decrease to approximately 0.36 alarms/hour. Considering that the false alarm rate achieved by DAICS was 1.96 alarms/hour with $W_{grace} = 5s$, SiFA decreases this rate by 81%. The FPR drops to 0.07%, indicating a significant decrease in the number of false alarms reported to the technician. On the other hand, the FNR increases from 14.63% to 15.43% due to a small number of malicious samples that are misclassified as benign, possibly because they are similar to historical false alarms. The time performance of the proposed algorithms is also measured. Algorithm 6 requires at most 21 ms to update the buffer and calculate the updated similarity threshold λ^g. Algorithm 3 requires less than $1ms$ to decide if there is a match between an alarm and historical alarms in the buffer.

The sensitivity of SiFA to the maximum buffer length L_{buffer} is also measured. As shown in Table 4.2, with the default value of $L_{buffer} = 177$, SiFA reports 0.3646 alarm/hour. Furthermore, decreasing L_{buffer} below 177 slightly improves the classification accuracy. However, it caused a decrease in the number of detected attacks to 29 and 30 in the case of $L_{buffer} = 100$ and 150 respectively. This is because decreasing the maximum buffer length increases the diversity of alarms in the buffer, i.e., the alarms are very dissimilar and have large distances between each other. As a result, the average minimum distance between these alarms increases, increasing the similarity threshold λ^g and causing more alarms to be suppressed, thus, some real attacks might also be suppressed.

Table 4.2: Evaluation of SiFA on the SWaT dataset

	Precision	Recall	FNR	FPR	F1	False Alarms (Per hour)	Detected Attacks	Missing Attacks
DAICS	0.9185	0.8616	0.1387	0.0115	0.8892	6.6	33	4, 13, 14
DAICS ($W_{grace} = 5s$)	0.9242	0.8525	0.1476	0.0105	0.8868	1.96	33	4, 13, 14
Buffer size = 100	0.9961	0.8536	0.1463	0.0005	0.9194	0.2917	30	4, 7, 13, 14, 21, 29
Buffer size = 150	0.9946	0.8474	0.1525	0.0007	0.09151	0.375	29	4, 7, 13, 14, 15 , 21, 29
Buffer size = 177	**0.9942**	**0.8456**	**0.1543**	**0.0007**	**0.9139**	**0.3646**	**31**	**4, 7, 13, 14, 29**
Buffer size = 200	0.9935	0.8462	0.1537	0.0008	0.9139	0.3854	31	4, 7, 13, 14, 29
Buffer size = 250	0.9935	0.8462	0.1537	0.0008	0.9139	0.3854	31	4, 7, 13, 14, 29

By contrast, increasing the maximum buffer length (e.g., $L_{buffer} = 200, 250$) decreases the diversity of the alarms in the buffer, which decrease the average minimum distance between these alarms, hence, the similarity threshold λ decreases. As a result, more alarms are reported to the technician, which may increase the number of false alarms. Furthermore, with $L_{buffer} = 200$, the number of false alarms increases slightly to 0.3854 alarm/h. Based on these results, it is recommended to use an L_{buffer} similar to the number of false alarms on a representative validation dataset.

Missing attacks SiFA missed five attack scenarios, including #4, #7, #13, #14 and #29. In attack #4, the attacker opens the motorised valve MV504 with the goal of halting the shut-down sequence of the Reverse Osmosis (RO) unit and reducing the life of the RO unit. As denoted in the dataset documentation, this attack was unsuccessful, and it did not cause any notable changes in the ICS. Attack #13 was carried out by closing the motorised valve MV304; however, the dataset records analysis revealed that this valve was already closed before the start of the attack, causing the attack to produce no effect.

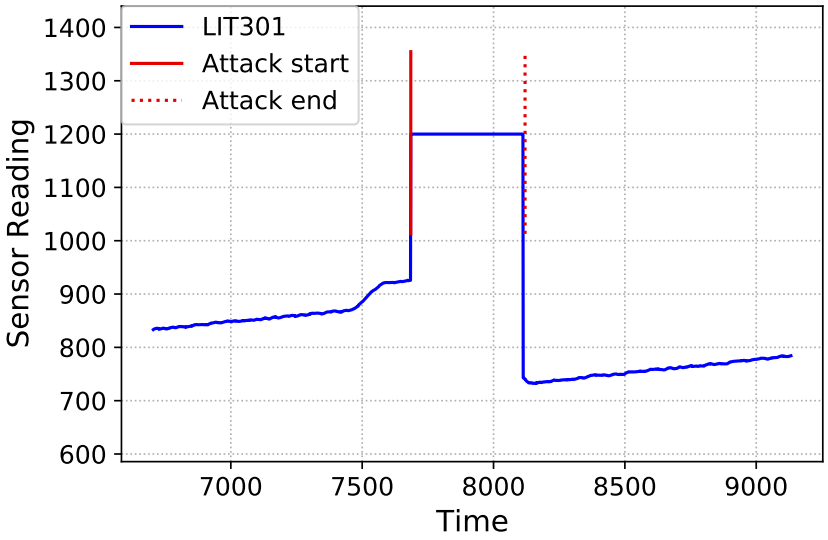

Figure 4.3: Reading of the water level sensor LIT301 before and after attack scenario #7.

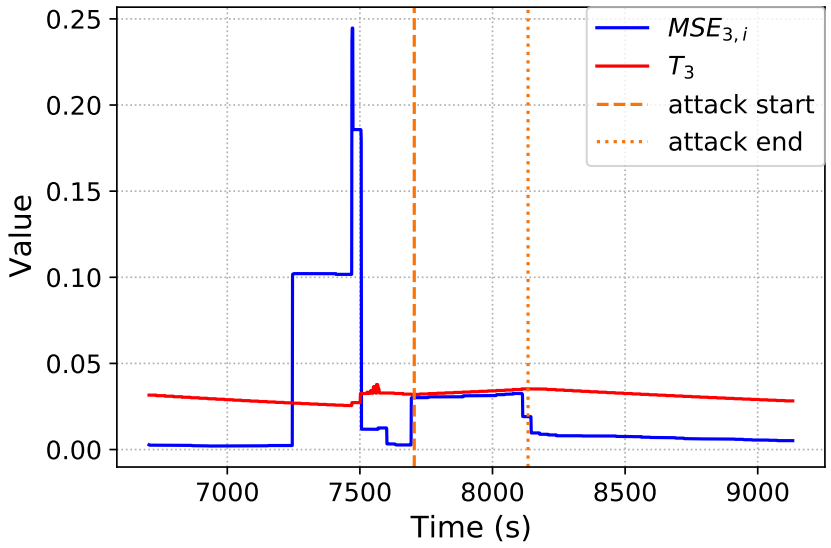

Figure 4.4: Prediction error during attack scenario #7. As shown, the prediction error is barely below the threshold.

In attack #7, the attacker spoofs the reading of sensor LIT301 and reports readings that are higher than reality, as shown in Figure 4.3. The goal is to stop the inflow of tank T301, causing tank underflow and damaging pump P301. This attack was detected using the original DAICS framework; however, it disappears after employing SiFA. During this attack, the mean square error $MSE_{3,i}$ of the third output section was just below the detection threshold T_3, as shown in Figure 4.4. This is because the third output section of WDNN was tuned on false alarms that are highly similar to this attack scenario; therefore, the WDNN produced a prediction error below the threshold.

Attack #14 was performed by keeping the motorised valve MV303 closed for eight minutes when the valve was supposed to be open. The goal of the attack was to halt stage 3 of the SWaT testbed. However, as mentioned in the dataset documentation, the attack failed because tank T301 was already full. Figure 4.5 confirms the stable water level in the tank during the attack. Furthermore, as illustrated in Figure 4.6, the prediction error is

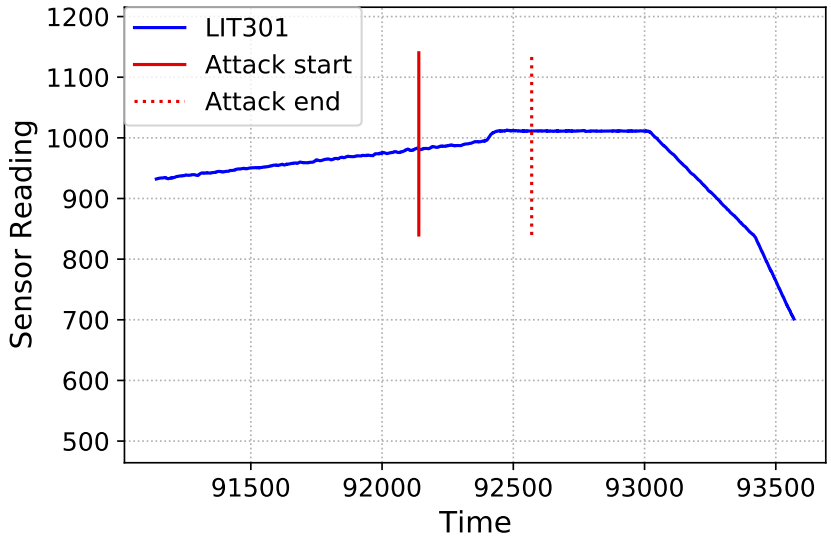

Figure 4.5: Reading of the water level sensor LIT301 before and after attack scenario #14.

well below the threshold.

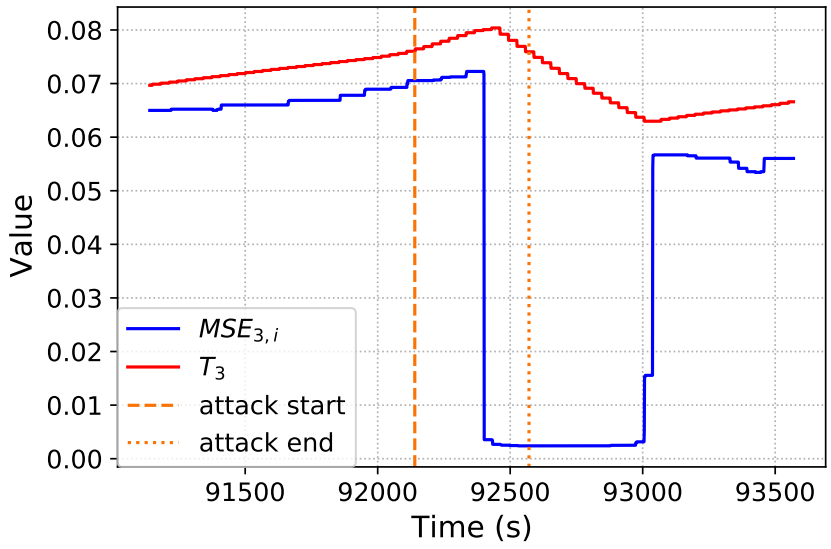

Figure 4.6: Prediction error during attack scenario #14. As shown, the prediction error does not exceed the threshold.

Attack # 29 was performed by turning on three pumps (P201, P203, P205) in order to waste chemicals. However, as noted in the dataset documentation, these pumps did not start because of a mechanical interlock,

and thus the attack failed. This attack was detected using DAICS but only after the attackers terminated the attack. However, after using SiFA, the alarm caused by this attack was classified as false and suppressed because it was similar to other historical alarms in the second output section. Given that the attack was unsuccessful, ignoring the alarm is reasonable.

Overall, SiFA improved the performance of DAICS on the SWaT dataset in terms of the F1 score and number of human interventions required due to false alarms. However, because of this, one attack (scenario #7) is missed. Further analysis of this attack was conducted using a counterfactual explanation. A counterfactual explanation of a prediction is the smallest change in features that can force the model to change the prediction to a predefined output [107]. In the case of attack #7, one sensor was compromised, and thus only the readings of that sensor can be changed. As shown in Figure 4.7, if LIT301 had increased by 10%, attack #7 would have always been detected.

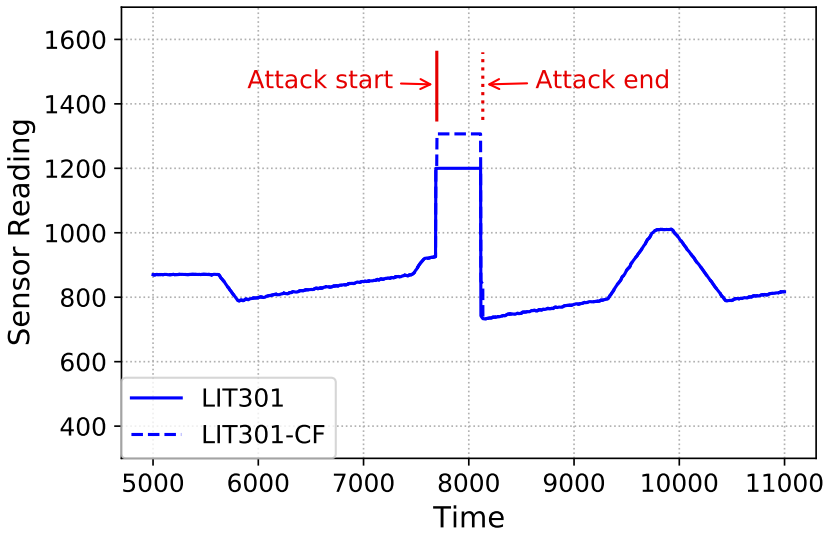

Figure 4.7: Counterfactual explanation of attack scenario #7. An increase of 10% in the sensor level during attack was sufficient to achieve a successful detection by SiFA.

4.4 Summary

In this chapter, we presented SiFA, an extension of DAICS for the reduction of false alarms in ADS monitoring ICS. SiFA collects a buffer of historical false alarms and suppresses every new alarm that is similar to these false alarms. The buffer is initialised with false alarms generated on a validation dataset, which comprises only benign data samples. This buffer is then updated (a new instance is added or an old instance is replaced) when the operator identifies a new false alarm. SiFA is evaluated using the dataset collected from the SWaT testbed. The evaluation results show that the SiFA can decrease the false alarm rate of DAICS by more than 80%. Furthermore, SiFA improves the detection accuracy of DAICS with an F1 score $> 91\%$. However, this comes at the cost of one missing attack. A counterfactual analysis of this attack demonstrated that an increase of 10% in the level of the LIT301 sensor during attack would have been sufficient to achieve a successful detection.

Chapter 5

Robustness to adversarial input perturbations

In the previous chapter, it was demonstrated that ML-based ADSs that are robust to non-adversarial input perturbations can be developed. In addition, it was shown that ML is an effective technique for detecting sophisticated cyberattacks that target CIs. However, ML methods can also be vulnerable to adversarial input perturbations, where an attacker attempts to fool the classification/prediction mechanism by crafting the input data.

In this chapter, we study the problem of vulnerability to adversarial input perturbations in the case of ML-based NIDSs that detect DDoS attacks. In such NIDSs, an attacker may use their knowledge of intrusion detection logic to generate malicious traffic that remains undetected. One way to solve this issue is to adopt adversarial training-based approaches, in which the training set is augmented with adversarial traffic samples. However, such approaches may employ adversarial samples that are generated based on target/victim ML models, causing the trained models to remain vulnerable to simple adversarial attacks [82]. Moreover, the majority of the proposed approaches are evaluated based on perturbations added to the extracted samples (i.e., perturbations on the feature space), which is impractical because it requires the attacker to bypass the feature extractor, as explained in Chapter 2.

These challenges can be addressed by answering two research questions:

- Can the generation process of adversarial samples used for training be decoupled from the target model?
- Can robust NIDSs be developed and evaluated in practical conditions?

This chapter answers the aforementioned questions by presenting an adversarial training framework for robust DDoS attack detection, namely GAN-based Adversarial training for robust DDoS attack deTection (GADoT), which leverages the GAN to generate adversarial DDoS samples for training. GANs are deep generative models that can learn the distribution of the input data, and produce similar new examples. A GAN consists of two models: a discriminator and a generator, both trained to compete with each other. The generator model captures the training data distribution to produce fake samples that resemble those of the training data. The discriminator model learns to distinguish between real and fake samples. An optimal generator causes the discriminator to fail in its task. The intuition here is that GAN-based sample generators can be exploited to enrich the training set with adversarial examples similar to those that an attacker would use to fool the NIDS, that is, DDoS examples with the feature distribution of benign samples.

GADoT is designed to produce strong adversarial examples, thus enabling the effective adversarial training of ML-based DDoS classifiers. The framework employs the generator model of a GAN trained on benign samples to produce fake-benign samples. Using these fake-benign samples, the DDoS samples are perturbed by replacing their features with values from those fake samples. The perturbed samples are then added to the training dataset, which already contains benign and unperturbed DDoS samples, creating an augmented dataset for adversarial training. The trained model is expected to classify DDoS samples correctly, irrespective of perturba-

tion. To the best of our knowledge, GADoT is the first solution that uses GAN-based perturbations to decouple the trained model from the process of generating adversarial samples to prevent the model from influencing the strength of adversarial samples [82]. Finally, practical evaluation and analysis are offered for the robustness of the ML models trained with GADoT using network traffic traces that capture adversarially perturbed SYN and HTTP DDoS flood attacks.

The rest of the chapter is structured as follows: Section 5.1 introduces the threat model of this work. The proposed GADoT approach is described in Section 5.2. In Section 5.3, the datasets, details of the specific feature perturbation and experimental design are presented. The results are discussed in Section 5.4. Finally, Section 5.5 summarises this chapter.

5.1 Threat Model

To characterise the adversary attacking our ML-based NIDS, we identify their goals, capabilities and knowledge, as per the threat modelling framework presented by Biggio et al. in [108], which is widely utilised to model the threats against ML models, e.g. [10, 109]. Our adversary model is summarised in Table 5.1.

5.1.1 Attacker's goal

The adversary aims to evade the ML model through manipulating the DDoS traffic to be misclassified as benign. If successful, this operation would allow the attack traffic to remain undetected and to reach the victim machine or network. This maps to the online evasion attack as defined in the recently published MITRE adversarial threat matrix [17].

Table 5.1: Adversary model

Threat Model Characteristic	**Type**	**Adversary View**
Adversary Goals	Compromising integrity (Evasion)	✓
	Compromising availability	✗
	Compromising confidentiality	✗
Adversary Knowledge	Feature set	✓
	Satisfy domain constraints	✓
Adversary Capabilities	Manipulate training data	✗
	Manipulate test data	✓
	Manipulate model	✗

5.1.2 Attacker's knowledge

The attackers know that the network traffic is monitored by an ML-based NIDS. However, they have no direct access to the NIDS itself or the underlying ML model. As a result, they cannot compromise the NIDS by bypassing some of its components (such as the feature extractor) in order to feed manipulated samples directly to the model. Thus, state-of-the-art attacks [68, 74, 75] based on techniques such as FGSM, which manipulate the data samples, are not viable in such an attack scenario. However, since DDoS attacks are well researched in the literature, the attackers can exploit their field knowledge to know the basic traffic features employed for attack detection. Based on that, they manipulate these features to evade the classification model. At the same time, the manipulated traffic must satisfy network domain constraints such that the objective of the attack remains valid. For example, a SYN Flood must be comprised of SYN packets. Similarly, the adversary should ensure that a manipulated packet will not be discarded for reasons such as invalid header fields (e.g., the checksum). These assumptions present a realistic threat model for the

NIDS problem space.

5.1.3 Attacker's capability

The attackers can perturb the DDoS flows at test time in order to evade the classification model, allowing the DDoS attack to pass undetected. They can also build upon the published adversarial attacks, such as [8], to select the traffic parameters that can be tuned to craft an evasion attack.

5.2 The GADoT approach

To produce a robust deep learning-based DDoS detector, we propose an adversarial training approach, GADoT, in which a ML model is trained with benign samples, DDoS samples, and adversarial DDoS samples. The goal is to increase the robustness against the perturbed DDoS samples, in anticipation of an attacker who can craft packets and perturb malicious flows to mimic those characteristics of benign flows. The approach employs a generator neural network that generates fake-benign samples used to create adversarial samples and a perturbation module that produces the augmented dataset T_{adv} used for adversarial training of the target model, as depicted in Figure 5.1.

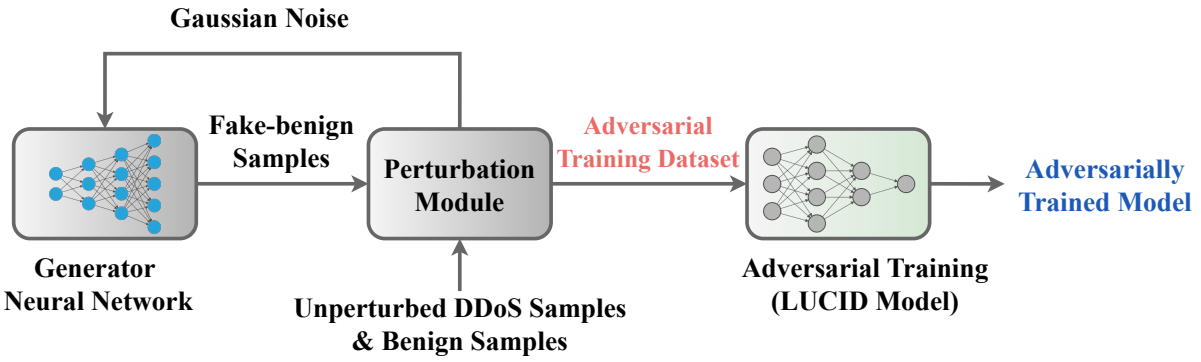

Figure 5.1: Illustration of the GADoT approach for adversarial training.

In the following, we briefly introduce the target model used in this work, LUCID[1] [61], and how it prepares the data samples before classification, then we explain the modules employed by GADoT. Finally, we describe two baseline approaches against which GADoT is compared.

5.2.1 LUCID as a target model

LUCID is an efficient DDoS detection system that implements a pre-processing tool for the network traffic and a CNN. The source code of both pre-processing tool and neural network are publicly available at [110]. The preprocessing tool, which we refer to as the feature extractor, converts the flows extracted from the network traffic into data samples compatible with the CNN. In Figure 5.2, we illustrate the feature extractor and the classification model. The figure also contains other details such as dataset names that we introduce later, in Section 5.3. The CNN classifies the data samples as either benign or DDoS. Each data sample includes a set of features extracted from each packet of a bi-directional flow in a specific time window.

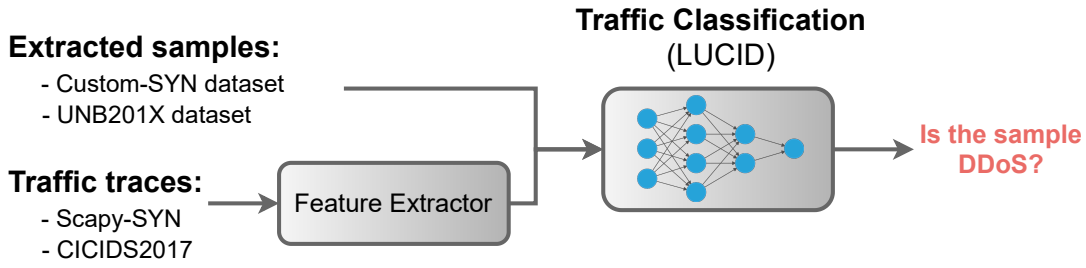

Figure 5.2: Illustration of the traffic classification process.

Using the LUCID feature extractor, we extract 11 features that are described in Table 5.2 and use ten seconds as the time window length. A

[1]We selected LUCID as our target model as it achieves the best classification accuracy of the deep learning-based DDoS detection solutions in the literature. Its preprocessing tool also enables us to adopt the practical approach of perturbing the network traffic.

network flow, or sample, is represented as an array of fixed size 10x11, whose 10 lines are the packets that belong to the flow collected within the time window, while the 11 columns are the features extracted from each packet. The size of the array is fixed and decided at training time, as this is a requirement for the CNN. Flows with more than 10 packets in a time window are truncated, while shorter flows are zero-padded. The number of actual packets in a sample (non zero-padded rows) can be seen as a further feature that we call *Flow length* in the rest of the chapter. An examples of a LUCID sample is presented in Table 5.3.

Table 5.2: The 11 packet header attributes used in LUCID

Feature	**Description**
Time	Relative time of each packet with respect to the first packet in the time window
Packet Len	Length of the entire IP packet
Highest Layer	The highest protocol in a packet, e.g. TCP or HTTP
IP Flags	Three bits taking one of three states: all zeros, do not fragment or more fragments
Protocols	Representing the protocols found in the packet
TCP Len	Size of the TCP segment (header + payload)
TCP Ack	Relative acknowledgement number
TCP Flags	Nine 1-bit flags of a TCP segment
TCP Win Size	The value of the field of the same name in a TCP segment
UDP Len	Size of the UDP segment (header + payload)
ICMP Type	The value of the type field of an ICMP message

5.2.2 The generator neural network

The generator neural network is the generator model $\mathcal{G}$ of a Wasserstein GAN with gradient penalty (WGAN-GP) [111] that achieves a state of the art performance in terms of training stability and diversity of the

Table 5.3: A sample after normalisation in the LUCID framework

	Pkt #	Time(sec)	Packet Len	Highest Layer	IP Flags	Protocols	TCP Len	TCP Ack	TCP Flag	TCP Window Size	UDP Len	ICMP Type
Packets	0	0.0	0.00156	0.786	0.0	0.235	4.86e-04	1e-08	0.0103	0.5965	0.0	0.0
	1	1e-05	0.001	0.786	1.0	0.235	0.0	2.4e-10	0.092	0.441	0.0	0.0
Padding	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	⋮	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

generated samples. In addition to the generator $\mathcal{G}$, the WGAN-GP includes a discriminator model $\mathcal{D}$ that learns to distinguish between the real samples from the training data and the generator's output. During the GAN training, $\mathcal{G}$ and $\mathcal{D}$ compete with each other. $\mathcal{G}$ learns to generate better fake samples that fool $\mathcal{D}$, whereas $\mathcal{D}$ learns to label those samples, minimising the success of $\mathcal{G}$. An optimal generator causes the discriminator to fail to distinguish between the real and generated samples.

Given a training dataset of benign and DDoS samples, we train the generator and discriminator of the WGAN-GP with the benign samples. After training, the discriminator is removed and the trained generator $\mathcal{G}$ of WGAN-GP becomes the generator of GADoT (Figure 5.1). This generator takes a source of noise z as an input and generates new samples $\hat{x} = \mathcal{G}(z)$ with the feature distribution of the training data. As WGAN-GP is trained with benign samples, the resulting $\mathcal{G}(z)$ produces what we call *fake* benign samples.

We chose to generate benign samples in order to perturb the features of the DDoS samples with values seen in benign traffic. This matches with the idea of DDoS flows disguised as benign through mimicking the characteristics of normal flows. For instance, [8, 10] show how the attacker can evade a wide range of classifiers after manipulating features such as the number of bytes and packets, inter-arrival time and packet rate in malicious network traffic. An advantage of using a GAN over the real

benign samples for perturbation is the wide variety of unique values that a GAN produces. A well-trained GAN produces a unique output for each random input, providing a wide range of fake-benign samples.

The architecture of the generator neural network is based on the deep convolutional generator of WGAN-GP [111] with minimal changes to match the shape of LUCID samples. The architecture is modified to have output shape 10x11, as per the shape of LUCID samples, and 20 as the dimension of the noise source z to keep it smaller than the dimensionality of the output [112]. To train this generator, we utilise the discriminator of the same WGAN-GP and modify its input shape to be 10x11. The other hyperparameters of both generator and discriminator are unchanged with respect to the original architecture. Finally, the output layer in [111] uses a *tanh* activation function that produces samples with features between [-1, 1]. We rescale the values between [0, 1] in accordance with the normalisation scheme adopted by LUCID, where all input features are normalised between 0 and 1 before classification. The architecture of both neural networks are given in tables 5.4 and 5.5.

Table 5.4: Architecture of the generator neural network

Generator $G(z)$		
	Kernel size	output shape
z	-	20
Dense, ReLU	-	$128 \times 6 \times 6$
Reshape	-	$6 \times 6 \times 128$
UpSampling2D	-	$12 \times 12 \times 128$
Conv2D	3×2	$10 \times 11 \times 128$
BatchNormalization, ReLU	-	$10 \times 11 \times 128$
Conv2D	4	$10 \times 11 \times 64$
BatchNormalization, ReLU	-	$10 \times 11 \times 64$
Conv2D, tanh	4	$10 \times 11x1$

Table 5.5: Architecture of the discriminator neural network

Discriminator $D(x)$		
	Kernel size	Output shape
Conv2D, LeakyReLU	3	$5 \times 6 \times 16$
Dropout(0.25)		$5 \times 6 \times 16$
Conv2D	3	$3 \times 3 \times 32$
ZeroPadding2D	-	$4 \times 4 \times 32$
BatchNormalization, LeakyReLU	-	$4 \times 4 \times 32$
Dropout(0.25)	-	$4 \times 4 \times 32$
Conv2D	3	$2 \times 2 \times 64$
BatchNormalization, LeakyReLU		$2 \times 2 \times 64$
Dropout(0.25)		$2 \times 2 \times 64$
Conv2D	3	$2 \times 2 \times 128$
BatchNormalization, LeakyReLU		$2 \times 2 \times 128$
Dropout(0.25)		$2 \times 2 \times 128$
Flatten		512
Linear		1

5.2.3 The perturbation module

The perturbation module takes a dataset with benign and DDoS samples, and fake-benign samples produced by the generator neural network and outputs the adversarial training dataset. The perturbed DDoS samples in the adversarial training dataset are crafted either by replacing features with values seen in benign traffic or by replacing the zero-padded rows with dummy packets. Both the dummy packets and the substitute values of features are obtained using the generator model.

Algorithm 8 shows the process of generating the adversarial training dataset T_{adv}. The inputs of the algorithm are a dataset T with both benign and unperturbed DDoS samples, the ground truth Y of the dataset and the list of features to perturb F, while the output is the adversarial training dataset T_{adv}. F is a subset of the 12 features of a sample, 11 packet header features plus *Flow length*, as described in Section 5.2.1. Note that each element in Y is a binary label $\in \{0, 1\}$, where 1 denotes a DDoS sample,

Algorithm 8 Adversarial dataset generation algorithm

Input: T: Training dataset; Y: True labels of T; F: List of features to perturb;
Output: T_{adv}: Adversarial dataset;

1: $T_{adv} = Copy(T)$; ▷ Initialise T_{adv}
2: **for** f **in** F **do**
3: $z = \mathcal{N}(0, 1)$; ▷ Source of noise
4: $T_g = \mathcal{G}(z)$; ▷ Generate fake-benign samples
5: $T_c = Copy(T)$;
6: **if** f == flow_length **then**
7: Fill zero-padded rows in $T_c[Y == 1]$;
8: **else**
9: $T_c[Y == 1, f] = T_g[:, f]$;
10: **end if**
11: $T_{adv}.append(T_c)$;
12: **end for**
13: **return** T_{adv}

and 0 is a benign sample.

In the algorithm, a copy of T initialises T_{adv} (line 1). To perturb a feature $f \in F$, we start with feeding a source of noise into the generator model $\mathcal{G}(z)$ to generate a set of fake-benign samples T_g; then we make a copy of T named T_c (lines 3–5). For f the *Flow length* feature, the goal is to perturb the number of packets in each DDoS sample of T_c; and thus the zero-padded rows in each sample are filled with packets from the fake samples of T_g (line 7). For the other features, the value of feature f in each DDoS sample of T_c is replaced by a value from the fake samples of T_g (line 9). Then T_c is appended to T_{adv} (line 11). In this way, T_{adv} is augmented with adversarial DDoS samples whose features can be observed in benign traffic.

The resulting T_{adv} is unbalanced, with more malicious than benign samples. To obtain a balanced dataset, we duplicate some of the benign samples until we reach an equal number of malicious and benign samples. Of course, the ground truth Y is updated with the labels of the newly added adversarial and benign samples. It is important to note that, in

contrast to the related work [75, 83], GADoT decouples the trained model from the generation of the adversarial samples used for training. This is important to prevent the trained model from influencing the strength of the generated adversarial samples [82].

5.2.4 Adversarial training

We use an adversarial training dataset T_{adv} in order to train the target model to increase its robustness to adversarially perturbed DDoS attacks. As T_{adv} contains benign samples as well as perturbed and unperturbed DDoS samples, the goal of training is to minimise the error between the ground truth and the predictions through optimising the learnable parameters of the model.

Having presented the modules of GADoT, we revisit Figure 5.1 to summarise the end-to-end approach. The input for the perturbation module is a labelled training dataset T, which contains unperturbed DDoS samples and benign samples. Then, the generator neural network receives a number of noise samples from the Gaussian noise source $z = \mathcal{N}(0, 1)$ to generate an equal number of fake-benign samples T_g to perturb the DDoS samples, as explained in Algorithm 8. This process produces the adversarial training dataset T_{adv}, which is then used to train the model, LUCID. The outcome of the GADoT approach is a robust DDoS detection model capable of distinguishing between benign samples and DDoS samples even under adversarial attack.

5.2.5 Baseline approaches

GADoT is compared against four baseline approaches that can generate perturbed DDoS samples for adversarial training. In the first approach, which we refer to as Benign Feature Perturbation (BFP), the training DDoS

samples are perturbed with features obtained from the benign samples in the training dataset. We use the same procedure explained in Algorithm 8 but with T_g from the training data to save the cost of the GAN. The second approach is similar to the first approach except that T_g is generated by a Pseudo Random Number Generator (PRNG) with uniform distribution $\mathcal{U}\{0, 1\}$. The third approach is based on the renowned FGSM [66] algorithm that uses the gradients of the neural network to create adversarial samples. We use FGSM with ℓ_∞ bound on the perturbation magnitude to create adversarial samples with minimal perturbations to fool the model. The FGSM obtains the adversarial examples by the following equation:

$$x^* = x + \epsilon \cdot sgn(\nabla_x L(f_\theta(x), y))$$

where $f_\theta(x)$ is the neural network parameters, x is the original example, $L(\cdot)$ is the cross entropy loss function, y is the true label of x, sgn denotes the sign function, $\nabla_x L(\cdot)$ is the gradients of the loss function with respect to the input x, ϵ is the allowed perturbation size, and x^* is the adversarial example.

The final approach uses the Projected Gradient Descent (PGD) algorithm to generate adversarial samples. PGD is a multi-step variant of FGSM, i.e., it iteratively applies FGSM to find the minimum perturbation sufficient to alter the model's decision. Therefore, PGD starts with an initialisation $x^0 = x$ and computes the adversarial example at the t-th step as:

$$x^t = \prod_{x+\mathcal{S}} \left(x^{t-1} + \alpha \cdot sgn\big(\nabla_x L\big(f_\theta\big(x^{t-1}\big), y\big)\big)\right)$$

where $\prod_{x+\mathcal{S}}$ is the projection of perturbation into the set $\mathcal{S}$, which is the set constraint by l_∞, and α is the step size.

For all baselines, the augmented dataset T_{adv} contains benign samples, DDoS samples, and adversarial DDoS samples with perturbed features as

well as perturbed padding rows.

5.3 Design of experiments

This section presents the datasets, the process of training different ML models, and the design of the experiments used to evaluate the performance of the GADoT approach in normal and adversarial settings.

5.3.1 Datasets

For our evaluation, we consider two types of DDoS attacks: TCP SYN flood [84] and HTTP GET flood [85]. Both aim at exhausting the victim's resources by sending large amounts of requests to the victim server, making its services unavailable to the users. We use four datasets: Custom-SYN and Scapy-SYN contain TCP SYN flood attacks while the CICIDS2017 [63] and the UNB201X [61] contain HTTP GET flood attacks. An overview of these datasets is provided in Table 5.6.

Table 5.6: Overview of the datasets

	Dataset	#Samples	#Benign	#DDoS
Training	Custom-SYN	52698	26349	26349
	UNB201X	265902	132746	133156
Evaluation	Scapy-SYN	18830	–	18830
	CICIDS2017	13193	–	13193

Custom-SYN dataset

A traffic trace recorded in our testbed. It combines legitimate traffic, e.g. web browsing and ssh, and SYN flood attack traffic generated by using the Hping network tool [113].

UNB201X dataset

This dataset was introduced in [61] to enable LUCID to better detect DDoS attacks irrespective of the training dataset. UNB201X is a balanced combination of preprocessed traces from ISCX2012 [114], CICIDS2017 and CSECIC2018 [63][2].

Scapy-SYN dataset

This dataset contains a SYN flood attack crafted by using the Scapy tool [115]. Scapy is an open source program to send, capture, dissect and forge network packets. It comprises seven different traffic traces: one with unperturbed SYN attack packets and six traces that contain SYN packets with different perturbed features:

(i) *IP flags*: bit *do not fragment* randomly set to 0 or 1.

(ii) *TCP length*: SYN packets with random payload content.

(iii) *Padding replacement*: the SYN packet is followed by a random number of dummy packets with a random delay.

(iv) *SYN Packet Replication*[3]: the SYN packet is repeated a random number of times with a variable delay.

(v) *IP flags & TCP length & Padding replacement*: combination of these three perturbations.

(vi) *IP flags & TCP length & SYN Packet Replication*: combination of these three perturbations.

With regard to *Padding replacement* and *SYN Packet Replication*, these are perturbations to the *Flow length* feature in which the attacker exploits

[2]The UNB201X dataset has kindly been made available by the authors of [61] to enable our evaluation.

[3]The nature of the TCP SYN flood attack with a single SYN packet constituting a flow makes this attack well suited for experimenting with perturbations on the *Flow length* feature.

the zero-padded rows in DDoS samples to fool the DDoS detection model. The attacker's aim is to force the feature extractor to fill them with a random number of additional packets, making the DDoS samples less dissimilar to the benign samples (in terms of number of packets per flow). This can easily be achieved by injecting dummy packets into the network with the same IP address and TCP port of the SYN packet, or by replicating the same SYN packet multiple times within a time window.

CICIDS2017 dataset

The CICIDS2017 dataset contains a wide range of normal and malicious network activities, including DDoS. We use the timeslot 3.30PM-5.00PM of the CICIDS2017-Fri7PM trace that contains HTTP DDoS traffic generated by using Low Orbit Ion Cannon (LOIC) [116]. We perturb the CICIDS2017 traffic trace with Scapy to introduce two perturbations: random delays between the DDoS packets and the fragmentation of those packets. The delay between packets is reflected in the packet arrival time, which is one of the features used by LUCID (see Table 5.2). Similarly, packet fragmentation decreases the *TCP Len* of each packet and thus increases the number of packets in the attack flows. Both perturbations are particularly suitable for the HTTP DDoS traffic, as each attack flow comprises several packets.

LOIC dataset

LOIC is an open-source network stress testing tool that can carry out a variety of DDoS attacks [116]. We deployed LOIC on a Windows virtual machine and carried out an HTTP GET flood attack on an Ubuntu virtual machine running a web server. LOIC was configured to terminate the connection with the web server after sending the HTTP-GET request without waiting for a reply. Meanwhile, Wireshark was used to sniff the packets traversing the network between the two virtual machines to create

Table 5.7: Summary of the methodology for training and evaluation of the LUCID models

Model Name	**Model Detection Goal**	**Training**	**Evaluation**
Model-SYN	SYN flood	Custom-SYN training dataset	Traffic traces: Scapy-SYN Perturbed Features: *IP Flags; TCP Len; Flow length* Crafting tool: Scapy
Model-HTTP	HTTP GET flood	UNB201X training dataset	Traffic traces: CICIDS2017 Perturbed Features: *Time TCP Len; Flow length* Crafting tool: Scapy

the network trace used in the experiments. Note that this trace is different from the CICIDS2017 trace in that a Windows machine was used as the attacker's host, while in the CICIDS2017 dataset, the attacker's host was an Ubuntu machine. The goal of generating this trace is to investigate whether a trivial change like using a different operating system on the attacker side can produce an adversarial attack.

5.3.2 Training and evaluating the models

We produce two separate models. A model called *Model-SYN* is trained to detect the SYN flood attacks, while a second model called *Model-HTTP* is trained to detect the HTTP GET flood attacks[4]. Both models are based on the LUCID code available at [110], hence implemented in Python v3.6.8 using the Keras API v2.2.4 on top of Tensorflow 1.13.1 [117] and the ADAM optimisation algorithm. In the following, we present the details of training and evaluating each model, with and without GADoT. A summary is provided in Table 5.7.

[4]This enables comparison with the original LUCID paper when using the UNB201X dataset.

Model-SYN

First, this model is trained with the Custom-SYN training dataset in order to detect the SYN flood attacks in normal settings, i.e. to classify unperturbed flood attacks. The model architecture is presented in Table 5.8. Using GADoT, the same model is adversarially trained with the dataset T_{adv} generated by inserting the Custom-SYN training set as input to Algorithm 8. As T_{adv} contains adversarial samples perturbed using the GAN neural network, we expect the model to be robust and correctly classify the SYN flood traffic in adversarial settings. This model is evaluated on the six perturbed traces of the Scapy-SYN dataset to assess the robustness after using GADoT.

Table 5.8: Architecture of Model-SYN neural network

Model-SYN			
	Kernels	Kernel size	Output shape
Conv2D, ReLU	1	3×11	$8 \times 1 \times 1$
MaxPooling2D	-	8×1	$1 \times 1 \times 1$
Flatten	-	-	1
Dense, Sigmoid	-	-	1

Model-HTTP

To detect HTTP GET flood attacks, we train this model with the UNB201X training dataset. The model architecture is given in Table 5.9. The trained model is used for attack detection in normal settings, i.e. to classify unperturbed flood attacks. Using GADoT, the model is retrained with the dataset T_{adv} generated by using the UNB201X training set as input to Algorithm 8. In both cases T_{adv} is generated based on F that includes the 11 features of LUCID samples as well as perturbed padding rows. We evaluate this model on the manipulated CICIDS2017 traffic traces. This provides an evaluation based on the practical adversary capability of perturbing

network traffic. The results of these experiments are presented in sections 5.4.1 and 5.4.2.

Table 5.9: Architecture of Model-HTTP neural network

Model-HTTP			
	Kernels	Kernel size	Output shape
Conv2D, ReLU	64	3×11	$8 \times 1 \times 64$
MaxPooling2D	-	8×1	$1 \times 1 \times 64$
Flatten	-	-	64
Dense, Sigmoid	-	-	1

In our experiments, the benign samples in the Custom-SYN and UNB201X test datasets are fixed throughout the experiments, while the DDoS samples in both datasets are replaced by a similar number of perturbed DDoS samples for the relevant experiment. In this way, variations in performance are solely attributed to the adversarially perturbed DDoS samples, and not to a change in benign samples.

5.3.3 Evaluation metrics

We evaluate the classification accuracy using the F1 score, which combines both the recall and the precision and hence provides an overall measure of the model's performance. The robustness can also be indicated by the FNR, which is the number of misclassified DDoS samples to the total number of DDoS samples in a test dataset. These metrics are formally defined as follows:

$$Pr = \frac{TP}{TP + FP} \quad Re = \frac{TP}{TP + FN}$$
$$FNR = \frac{FN}{FN + TP} \quad F1 = 2 \cdot \frac{Pr \cdot Re}{Pr + Re}$$

where *Pr=Precision*, *Re=Recall*, *F1=F1 Score*, *FNR=False Negative Rate*,

TP=True Positives, *FP=False Positives*, *FN=False Negatives*.

5.4 Experimental Results

This section presents the detection results of the models trained using the baseline approaches and our adversarial training approach. We employ these models to classify network traces with perturbed DDoS traffic. Moreover, it is important to stress that the GAN is only used to generate adversarial samples for training. In the following, we use the difference Δ in the evaluation metric, e.g. F1 score or FNR, to show the effect of using GADoT on the robustness of the trained models. This difference is formally defined as: $\Delta = Result(After) - Result(Before)$.

5.4.1 Evaluation in normal settings

First, we want to measure any negative effect of adversarial training with GADoT on LUCID when tested on unperturbed data. Thus, we want to ensure that training with adversarial samples does not cause any degradation to the performance of LUCID. In this regard, Table 5.10 shows the detection results of the trained models on unperturbed test datasets before and after using GADoT. The results after using the baseline approaches are almost identical to those of GADoT and therefore omitted from the table for printing convenience.

Before using GADoT, the LUCID models are capable of detecting DDoS flows with F1 scores above 99%, which aligns with the results already reported in [61]. After training with each of the baseline approaches, the models also maintain their detection accuracy with F1 scores above 99% in all datasets. When trained with GADoT, the models maintain high accuracy scores, despite a drop in the F1 score of less than 1.5%. Notably, we observe a slight drop in precision on the SYN datasets, and in recall

Table 5.10: Evaluation of Model-SYN and Model-HTTP against unperturbed test datasets before and after using GADoT

Dataset (Model Name)	**Scapy-SYN (Model-SYN)**			**CICIDS2017 (Model-HTTP)**		
	Before	**After**	**Δ**	**Before**	**After**	**Δ**
Precision	0.9985	0.9739	-0.0246	0.9873	0.9902	0.0029
Recall	1.0000	1.0000	0.0000	0.9990	0.9978	-0.0012
F1 score	0.9992	0.9867	-0.0125	0.9931	0.9940	0.0009
FNR	0.0000	0.0000	0.0000	0.0009	0.0021	0.0012

score on the CICIDS2017 dataset. This indicates that a few ambiguous network flows are misclassified, which matches the observations made in [77] after adversarial training and [109] after employing ensemble learning to improve robustness.

5.4.2 Evaluation in adversarial settings

In what follows, we test Model-SYN and Model-HTTP on perturbed flood attacks that can be generated according to the attacker's knowledge as defined in our threat model (Section 5.1). Moreover, we demonstrate the positive impact of using GADoT on the models' robustness.

Model-SYN

The F1 scores of Model-SYN before and after using GADoT for training, and tested on perturbed samples, are shown in Table 5.11.

With no adversarial training, we observe a substantial sensitivity of the model to adversarial SYN flood attacks built with perturbations on *IP flags*, and *Padding replacement*, which leads to a drop in the F1 score to 65.96%, and 78.75%, respectively. The results obtained with *IP flags* confirm the high ranking of this feature in the analysis presented in the

original LUCID paper [61]. On the other hand, the impact of manipulation of the *Flow length* feature through *Padding replacement* can be explained by analysing the properties of the SYN flood attack traffic. Indeed, the *Flow length* of the SYN samples is either one or two packets (the SYN packet of the attacker and the SYN-ACK from the victim server, if its connection tables are not full), while benign samples in our datasets are longer on average. This difference becomes less prominent when the *Flow length* feature of DDoS samples are perturbed with *Padding replacement*, leading to a decrease in the classification accuracy.

These results demonstrate that perturbing the *IP flags* and replacing the padding rows with dummy packets can generate a successful evasion attack. We also identify that similar success can be achieved by perturbing a combination of features in the same SYN network flow. For instance, while introducing *SYN Packet Replication* perturbation has decreased the F1 score to 96%, combining it with the *IP flags* and *TCP length* perturbations yields a further 33% F1 score decrease, producing a stronger evasion attack.

We also observe that adversarial training might produce a decrease in the classification accuracy when perturbed samples are present. This is indicated by the negative Δ values in Table 5.11. Although minimal, the decrease is caused by the change in the distribution of the features due to the adversarial samples added to the training set. This change moves the decision boundary, and hence the accuracy scores, F1 included.

Adversarial training with the BFP baseline approach allows the Model-SYN to restore most of its detection accuracy with the majority of F1 scores $> 97\%$ except for the *IP Flags* perturbation. This indicates that adversarial training with this approach leaves the model vulnerable to perturbation of features not present in the benign samples. Similarly, adversarial training with the PRNG leads to an improvement of robustness against most of the perturbations except for the *IP flags*. The other two baseline approaches,

based on FGSM and PGD, lead to a minor improvement in robustness against most of the perturbations. This minor increase in robustness suggests that these baseline approaches might not be the best option to generate adversarial examples for the training of NIDS. On the other hand, after retraining using GADoT, we obtain an overall higher classification accuracy ($> 98\%$) and a substantial reduction in the false negative rate, which drops to reach approximately 0% for all perturbations. The results highlight that the adversarial dataset T_{adv} created with GADoT covers the space of adversarial perturbations that attackers might exploit. This also indicates that Model-SYN is more robust to adversarial attacks with adversarial training using GADoT than without.

Model-HTTP

As noted in Section 5.2.1, packet arrival *time* and *TCP Len* are important features in the detection model. We therefore measure the impact of delay perturbation and packet fragmentation on the detection accuracy. Packet fragmentation perturbs the *TCP Len* feature of each packet and increases the number of packets in each flow as well. Indeed, packet arrival time, packet size and the number of packets in a flow are features employed to generate adversarial network flows across the state-of-the-art, e.g., [8, 71–73, 77]. We apply the perturbations to the DDoS packets of the HTTP-GET flows in the CICIDS2017 dataset. Moreover, we also generated a LOIC network trace in which the attacker uses a host running a Windows operating system instead of the Ubuntu machine used in the CICIDS2017 dataset.

Table 5.11: Evaluation of Model-SYN against perturbed traces of the Scapy-SYN dataset

Perturbations	Before		GADoT				BFP				PRNG			
	F1 Score	**FNR**	**F1 Score**	**Δ**	**FNR**	**Δ**	**F1 Score**	**Δ**	**FNR**	**Δ**	**F1 Score**	**Δ**	**FNR**	**Δ**
IP Flags	**0.6596**	**0.5071**	**0.9867**	**0.3271**	**0.0003**	**-0.5068**	0.6558	-0.0038	0.5071	0.0000	0.6478	-0.0118	0.5071	0.0000
TCP Len	0.9992	0.0000	0.9867	-0.0125	0.0000	0.0000	0.9953	-0.0039	0.0000	0.0000	0.9862	-0.0130	0.0000	0.0000
SYN Packet Replication	0.9547	0.0851	0.9867	0.0320	0.0000	-0.0851	0.9951	0.0404	0.0000	-0.0851	0.9862	0.0315	0.0000	-0.0851
Padding Replacement	**0.7875**	**0.3494**	**0.9867**	**0.1992**	**0.0000**	**-0.3494**	**0.9757**	**0.1882**	**0.0382**	**0.3112**	**0.9862**	**0.1987**	**0.0000**	**-0.3494**
IP Flags; TCP Len; SYN Packet Replication	**0.6322**	**0.5368**	**0.9846**	**0.3524**	**0.0041**	**-0.5327**	**0.9888**	**0.3566**	**0.0130**	**-0.5238**	0.8060	0.1738	0.3059	-0.2309
IP Flags; TCP Len; Padding Replacement	**0.5862**	**0.5843**	**0.9867**	**0.4005**	**0.0000**	**-0.5843**	**0.9731**	**0.3869**	**0.0431**	**-0.5412**	**0.9262**	**0.3400**	**0.1130**	**-0.4713**

Perturbations	PGD				FGSM			
	F1 Score	**Δ**	**FNR**	**Δ**	**F1 Score**	**Δ**	**FNR**	**Δ**
IP Flags	0.6544	-0.0052	0.5074	0.0003	0.6541	-0.0055	0.5071	0.0000
TCP Len	0.9937	-0.0055	0.0000	0.0000	0.9933	-0.0059	0.0000	0.0000
SYN Packet Replication	0.9937	0.0390	0.0000	-0.0851	0.9933	0.0386	0.0000	-0.0851
Padding Replacement	0.8038	0.0163	0.3194	-0.0300	0.9127	0.1252	0.1491	-0.2003
IP Flags; TCP Len; SYN Packet Replication	0.8627	0.2305	0.2318	-0.3050	0.9066	0.2744	0.1596	-0.3772
IP Flags; TCP Len; Padding Replacement	0.5963	0.0101	0.5697	-0.0146	0.5999	0.0137	0.5657	-0.0186

Table 5.12: Evaluation of Model-HTTP against perturbed traces of the CICIDS2017 and LOIC datasets

Perturbations	Before		GADoT				BFP				PRNG			
	F1 Score	**FNR**	**F1 Score**	**Δ**	**FNR**	**Δ**	**F1 Score**	**Δ**	**FNR**	**Δ**	**F1 Score**	**Δ**	**FNR**	**Δ**
Delay	**0.5516**	**0.6150**	**0.9871**	**0.4355**	**0.0177**	**-0.5973**	0.9038	0.3522	0.1659	-0.4491	0. 8609	0.3093	0.2374	-0.3776
Packet Fragmentation	0.9406	0.1024	0.9935	0.0529	0.0000	-0.1024	0.9879	0.0473	0.0112	-0.0912	0.9909	0.0503	0.0093	-0.0931
LOIC	**0.0592**	**0.9689**	**0.9741**	**0.9149**	**0.0326**	**-0.9363**	**0.9744**	**0.9151**	**0.0314**	**-0.9375**	**0.9732**	**0.9140**	**0.0385**	**-0.9304**

Perturbations	PGD				FGSM			
	F1 Score	**Δ**	**FNR**	**Δ**	**F1 Score**	**Δ**	**FNR**	**Δ**
Delay	0.8086	0.2570	0.3155	-0.2995	0.8105	0.2589	0.3124	-0.3026
Packet Fragmentation	0.6388	-0.3018	0.5267	0.4243	0.8833	-0.0573	0.2019	0.0995
LOIC	**0.9769**	**0.9177**	**0.0314**	**-0.9375**	**0.9771**	**0.9179**	**0.0314**	**-0.9375**

As shown in Table 5.12, with no adversarial training, the HTTP GET attack in the CICIDS2017 dataset with a delay perturbation generates a strong evasion attack, as the FNR increases to 61.5%, confirming the importance of this feature for the detection model. On the contrary, packet fragmentation generates a weaker evasion attack with FNR around 10%. The reason is that the attack flows naturally comprise several packets, and hence packet fragmentation has a less noticeable impact on the attack flows. Moreover, the HTTP GET attack in the LOIC dataset generates another strong evasion attack, as the FNR increases to 96.89%, demonstrating that changing the operating system of the attacker's host can produce an adversarial attack. Further analysis showed that the LOIC trace contains *IP Flags* patterns different from those in the CICIDS2017 dataset, i.e., changing the operating system in which LOIC is deployed perturbs the *IP Flags* feature. This is aligned with the results of the Scapy-SYN dataset that show that perturbing the *IP Flags* generates a strong evasion attack.

With adversarial training using the baseline approaches, it is evident that the trained models remain vulnerable to DDoS attacks with perturbed samples. The better of the four approaches, which is the BFP, is still vulnerable to the delay perturbation as the F1 score and FNR are approximately 90% and 16% respectively. However, in the case of the LOIC dataset, training using the baseline approaches decreases the FNR to less than 4%. After using GADoT for training, the detection accuracy of the model in the case of delay perturbation experiences more than 43% improvement in the F1 score, reaching 98.71%. The same improvement is also apparent in the FNR that drops to 1.8%, indicating the increased resilience to adversarial perturbations. A similar level of resilience is also noticed in the case of the packet fragmentation and the LOIC dataset as the FNR drops to 0% and 3.26% respectively. This highlights that, in comparison with the baseline approaches, GADoT is able to create adversarial training dataset T_{adv} with

a wide coverage of adversarial perturbations. These results, combined with those of Model-SYN, demonstrate the considerable improvement in robustness after using GADoT.

5.4.3 Discussion

The results presented in the previous sections confirm that ML-based NIDSs are vulnerable to adversarial attacks. Potential adversaries can exploit their domain expertise to craft perturbed flood attacks without requiring knowledge of the underlying detection model. For example, they can perturb the delay in the network flows of the HTTP GET flood attack, as we have done in the CICIDS2017 dataset, which causes the FNR (misclassified DDoS samples) to reach 61.5%. Building upon this FNR, they can increase the number of bots (botnet DDoS attacks) to exhaust the victim resources and achieve a successful DDoS attack. In contrast, after adopting GADoT, most of the perturbed flood samples are detected, with the FNR reduced to below 1.8%.

The models trained using GADoT achieve high performance with F1 scores above 98% on perturbed DDoS network traces, improving on the 62% average detection rate reported in [73]. We highlight that the models trained using GADoT have detected the SYN and HTTP GET flood attacks, whether they are perturbed or not, allowing for the mitigation of such attacks. Moreover, the architecture of those models remains unchanged, and thus GADoT has no impact on their performance in terms of processed samples/second and memory requirements.

Finally, a consideration for practical deployment of a ML system such as GADoT is prediction interpretability [118]. We undertook a simple study of how our trained model classifies samples after adversarial training by exploring the filter activations of a convolution layer while processing samples. This revealed a change in the filter activation across different

features as well as a shift in their ranking, and hence their contribution to the model's decision. For example, our initial analysis indicated that there is no longer a heavy dependence on a single feature such as *Highest layer*, as reported in [61], which suggests to us that the adversarial training evens the activation across multiple features, which results in the increased robustness to evasion attacks. However, this is not conclusive and requires an in-depth analysis, which we propose for our future work.

5.5 Summary

Machine learning has been proven to be effective in NIDSs, demonstrating that DDoS attacks can be detected with high accuracy. However, ML methods can be vulnerable to adversarial input perturbations. The results of this study confirm that attackers can exploit their knowledge of the detection logic to generate adversarial traffic that evades the NIDS and reaches the victim machine to achieve the DoS goal.

To address this problem, this chapter introduces GADoT, which is an approach for adversarial training of ML models. Two LUCID models, which are state-of-the-art for DDoS detection, are further trained to detect two types of flood attacks, namely, SYN flood and HTTP GET flood. However, similar to other ML models, these models are vulnerable to adversarially perturbed DDoS samples and traces. To defend against such adversarial attacks, both models were trained using GADoT, which leverages GAN to generate adversarial DDoS samples for training. The GAN generates those fake samples independently from the victim ML model. Thus, the adversarial sample generation process is decoupled from the target/victim model, which allows the generation of strong adversarial samples for training.

The trained models were evaluated on real-world perturbed traces generated without knowledge of the victim model. This approach sets GADoT

apart from other similar solutions. The results show that the models trained with GADoT can detect both the SYN and HTTP flood attacks with high accuracy, regardless of whether they are perturbed. The models achieved an F1 score of more than 98% and the FNR dropped to less than 1.8% on perturbed DDoS network traces.

Chapter 6

UPAS: a universal attack against robust NIDSs

In the previous chapter, the GADoT approach was tested in scenarios that included binary classifiers (DDoS/benign). However, it is unknown whether the results obtained with binary classifiers generalise to multi-class classifiers aiming to detect various types of attacks, implying that the robustness of the models trained with GADoT may be overestimated. Therefore, there is a need for a standardised evaluation of adversarial robustness to avoid robustness overestimation, which may occur due to the use of weak adversarial attacks to evaluate the robustness of a proposed approach [119].

In this chapter, we take a step in the direction of evaluating the robustness of ML-based NIDSs that are resilient to adversarial perturbations. We propose the Universal Perturbation Attack against robust NIDSs (UPAS) that manipulates malicious network traffic in the traffic space to evade robust NIDSs. UPAS is a black-box attack because it requires no knowledge of the payloads of the packets, the detection ML model or the training set of the ML model. The manipulated attack traffic satisfies network domain constraints such that the objective of the attack remains valid and the traffic is not discarded for reasons such as invalid header fields (e.g., the checksum). For evaluation purposes, traffic from a real-world multi-attack

dataset is manipulated to inject the UPAS activity. The manipulated traffic is then used to evaluate the robustness of two state-of-the-art methods that are resilient to adversarial perturbations: a denoising autoencoder [73] and a DL model adversarially trained with GADoT [16].

The rest of the chapter is structured as follows: Section 6.1 introduces the adversary model of this study. The proposed attack is described in Section 6.2. The results are discussed in Section 6.3. Finally, Section 6.4 summarises the chapter.

6.1 Adversary model

6.1.1 Adversary's goal

The adversary aims to evade the robust NIDS through manipulating the malicious traffic so that it is misclassified as benign. If successful, this operation would allow the attack traffic to remain undetected and to reach the victim machine or network. This maps to an online evasion attack, as defined in the recently published MITRE adversarial threat matrix [17].

6.1.2 Adversary's knowledge

The attackers know that the network traffic is monitored by an ML-based NIDS. However, they have no direct access to the NIDS itself or the underlying ML model. As a result, they cannot compromise the NIDS by bypassing some of its components (such as the feature extractor) in order to feed manipulated samples directly to the model. However, since cyberattacks are well researched in the literature, the attackers can exploit their field knowledge to determine the basic traffic features employed for attack detection. Based on that, they manipulate these features to evade the NIDS. At the same time, the manipulated traffic must satisfy network

domain constraints such that the objective of the attack remains valid. For example, a SYN Flood must be comprised of SYN packets. Similarly, the adversary should ensure that a manipulated packet is not discarded for reasons such as invalid header fields (e.g., the checksum). These assumptions present a realistic threat model for the NIDS problem space.

6.1.3 Adversary's capability

The attackers can perturb the malicious flows at test time in order to evade the NIDS, allowing the cyberattack to pass undetected.

6.2 The UPAS attack

The UPAS attack evaluates the robustness of ML-based NIDSs designed to be resilient to adversarial perturbations. An adversarial perturbation is an imperceptible noise added by an attacker to the input of a ML classifier to alter the classification decision. In the case of NIDSs, attackers use their knowledge of the detection logic to inject perturbations into the network traffic, crafting adversarial traffic that remains undetected.

The UPAS attack injects a random time delay between 0 and 10 seconds before the packets sent by the attacker and thus perturbs the inter-arrival time between packets of a flow. This perturbation is particularly suitable for modifying the live network traffic of most network attacks, e.g., DDoS, brute force, infiltration and web attacks. Note that, this attack neither introduce bandwidth overhead nor affect statistical features, such as the number of flows between source and destination per second.

6.3 Experimental Evaluation

The UPAS attack is proposed to evaluate the robustness of NIDSs resilient to adversarial input perturbations. The attack activity is injected in a multi-attack dataset and used to evaluate the robustness of two robust NIDS. This section presents the the datasets, trained NIDSs, evaluation metric and experimental results.

6.3.1 Datasets

CICIDS2017 dataset

The CICIDS2017 dataset contains a wide range of normal and malicious network activities. For our study, the network traffic of eight attacks is extracted from the traffic traces of the CICIDS2017 dataset. The eight attacks are DoS, DDoS, FTP-Patator, SSH-Patator, Port Scan, Bot, Infiltration and Web-Attack. Then, the traffic of DoS and DDoS attacks are merged into one attack trace called the DoS trace. Thus, the NIDSs trained with this dataset detect seven attack classes in addition to a benign class. The benign class represents network traffic in an attack-free trace, i.e., the Monday trace in the published dataset. The dataset is balanced and split into 90% training and 10% test sets, with 10% of the training set used for validation.

UPAS dataset

The UPAS attack activity is injected into the attack traces of the CICIDS2017 dataset to create the UPAS dataset. Each attack trace is perturbed using Scapy [115] to add a random time delay between 0 and 10 seconds before the packets sent by the attacker.

6.3.2 Targeted NIDSs

Two NIDS are trained using the CICIDS2017 dataset. First, a LUCID model is trained using the GADoT adversarial training approach to detect seven attacks with benign traffic considered the eighth class. The architecture of the LUCID model is listed in Table 6.1. The trained model is expected to successfully detect the traffic from the eight classes, regardless of any perturbation. LUCID uses a feature extractor that extracts 11 features and uses 100 seconds as the time window length. These 11 features are described in Table 6.2. A network flow, or sample, is represented as an array of fixed size 100x11, whose 100 lines are the packets that belong to the flow collected within the time window, while the 11 columns are the features extracted from each packet. The size of the array is fixed and decided at training time, as this is a requirement for the CNN. Flows with more than 100 packets in a time window are truncated, while shorter flows are zero-padded. The LUCID model is trained with the CICIDS2017 training set using the GADoT approach and validated on the UPAS dataset.

Table 6.1: Architecture of LUCID neural network trained with GADoT

	Kernels	Kernel size	Output shape
Conv2D, ReLU	64	3×11	$8 \times 1 \times 64$
MaxPooling2D	-	8×1	$1 \times 1 \times 64$
Flatten	-	-	64
Dense, Softmax	-	-	8

Second, Reconstruction from Partial Observation plus (RePO+) is evaluated as an anomaly based NIDS that increases robustness by denoising the samples to remove adversarial perturbations [73]. RePO+ uses a 100 random mask to hide random parts of an input sample that is fully reconstructed by a denoising autoencoder. The usage of a random mask might hide the perturbed part of the sample, and hence, the autoencoder reconstructs a clean copy for anomaly detection. A sample is an array

Table 6.2: The 11 packet header attributes used in LUCID

Feature	**Description**
Time	Relative time of each packet with respect to the first packet in the time window
Packet Len	Length of the entire IP packet
Highest Layer	The highest protocol in a packet, e.g. TCP or HTTP
IP Flags	Three bits taking one of three states: all zeros, do not fragment or more fragments
Protocols	Representing the protocols found in the packet
TCP Len	Size of the TCP segment (header + payload)
TCP Ack	Relative acknowledgement number
TCP Flags	Nine 1-bit flags of a TCP segment
TCP Win Size	The value of the field of the same name in a TCP segment
UDP Len	Size of the UDP segment (header + payload)
ICMP Type	The value of the type field of an ICMP message

of statistical features extracted from each network flow using the feature extractor CICFlowMeter [64]. Given that RePO+ is an anomaly based NIDS, it is trained with the benign traffic of the CICIDS2017 dataset. RePO+ defines a score function that computes the error between a sample and its reconstructed version. The computed scores are compared to a predefined threshold, above which a sample is considered an anomaly. The threshold is selected based on the FPR, which is the ratio of benign samples that is considered malicious to the total number of benign samples when the threshold of the model is fixed to a specific number. Therefore, two thresholds are selected based on FPR = 0.01 and 0.1. The RePO+ version that uses FPR = 0.01 is called **RePO+A**, whereas the version with FPR=0.1 is called **RePO+B** in the rest of the chapter. The authors of RePO+ released their trained models publicly; hence, we use the published flow-based model in this study.

Both NIDSs are implemented in Python v3.6.8 using the Keras API

v2.2.4 on top of Tensorflow 1.14.0.

6.3.3 Evaluation metric

The robustness can be indicated by the FNR, which is the ratio of misclassified malicious samples to the total number of malicious samples in a test dataset. This metric is formally defined as follows:

$$FNR = \frac{FN}{FN + TP}$$

where *FNR=False Negative Rate*, *TP=True Positives*, *FN=False Negatives*.

6.3.4 Experimental results

The proposed adversarial attack is used to evaluate the real robustness of two NIDS designed to be robust against adversarial attacks. First, the results of the GADoT approach are presented followed by the results of the RePO+ NIDS.

The GADoT approach

To evaluate the robustness before adversarial training, a LUCID model is trained using the CICIDS2017 training set (without using GADoT) and evaluated on the UPAS dataset. As shown in Figure 6.1, the model is vulnerable to the UPAS adversarial attack with an FNR of approximately 90% in several attacks (blue bars). The model was then retrained using the GADoT adversarial training approach. After adversarial training, the model remains vulnerable to the adversarial attack, as the FNR is more than 20% in most attacks (orange bars). Specifically, in SSH-Patator and Bot attacks, GADoT fails to increase the robustness.

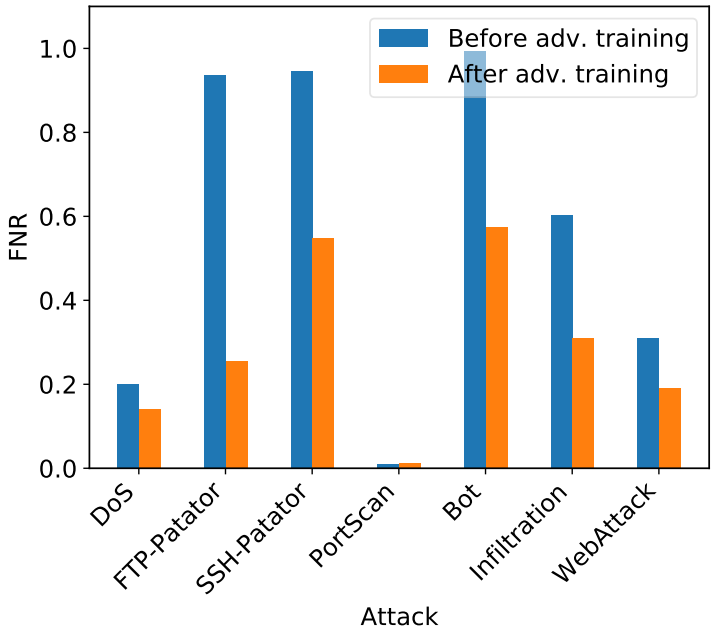

Figure 6.1: FNR of the model trained with GADoT for each attack when sending adversarial traffic.

The RePO+ NIDS

The performance of the RePO+ approach on the UPAS dataset is shown in Figure 6.2. As shown, in the case of RePO+A (FPR=0.01), the model fails to remove adversarial perturbation from a significant portion of adversarial traffic, and thus fails to detect perturbed network flows, as the FNR of most of the attacks is more than 50%. On the other hand, RePO+B (FPR=0.1) successfully detects the adversarially perturbed traffic of most of the attacks, except the SSH-Patator and Infiltration attacks, as the FNR is more than 40% in both the attacks. Recall that the threshold of RePO+B is lower than that of RePO+A, resulting in more false positives and more attacks detected.

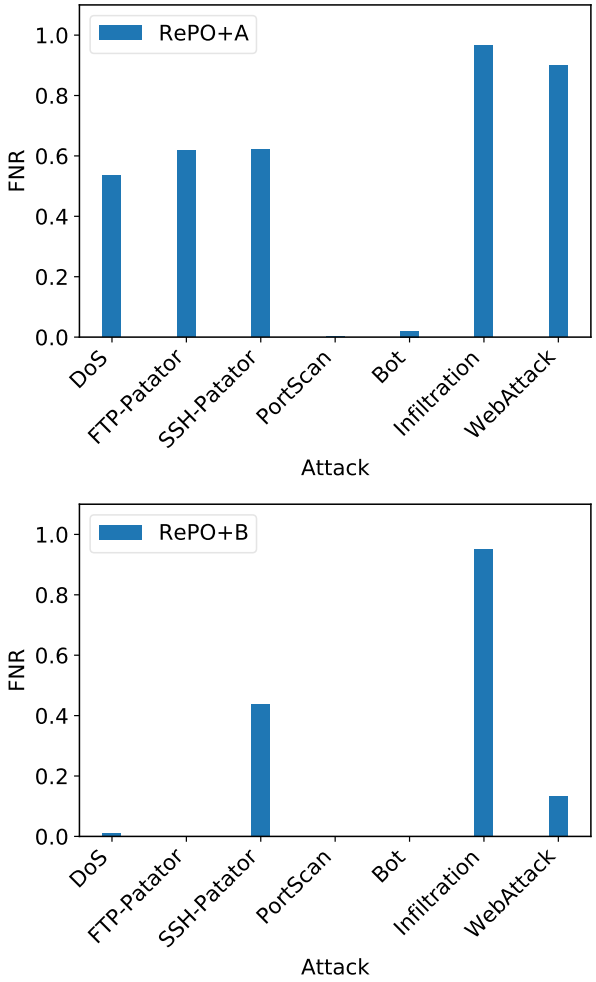

Figure 6.2: FNR of RePO+ for each attack when the FPR is 0.01 (above) and 0.1 (below) when sending adversarial traffic.

Defending against UPAS

We propose a simple adversarial training technique to defend against the proposed attack. This technique aims to create an adversarial training dataset to replace the GADoT approach in training the ML model without using a GAN. This adversarial training dataset comprises benign, malicious, and perturbed malicious samples. The perturbed malicious samples are created through splitting each malicious sample in the CICIDS2017 training dataset into three samples with a smaller number of packets.

Figure 6.3 shows the FNR of the model trained using the adversarial training dataset. The trained model yields better results than the GADoT and RePO+ approaches, as the FNR is less than 20% in most attacks. The only exception is the Bot attack in which approximately 50% of adversarial traffic evades the trained model. Furthermore, the FPR is less than 2%, which is comparable to the RePO+A model (FPR=1%) but with better results. Note that, this defence approach eliminates the overhead of the GAN, allowing it to be more computationally efficient than GADoT.

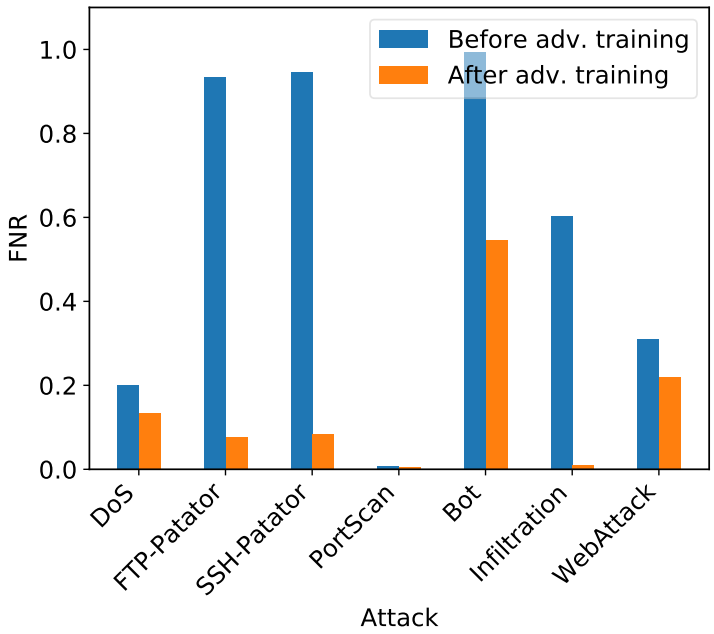

Figure 6.3: FNR of the defence technique for each attack when sending adversarial traffic.

6.3.5 Discussion

The results presented in the previous section demonstrate that the lack of a standardised robustness benchmark might lead to an overestimation of robustness. Most of the proposed NIDSs (e.g., RePO+ [73]) are evaluated against perturbations implemented in the feature space. We demonstrated that RePO+ is vulnerable to an adversarial attack generated in the traffic

space, namely, the UPAS attack. This indicates that researchers should evaluate the proposed robust solutions against traffic space perturbations to avoid overestimating robustness. On the other hand, GADoT, which is evaluated with traffic space perturbations, performs well in binary classification scenarios (DDoS/benign). However, the performance degrades in the scenario of multi-class classification. Therefore, the evaluation of the proposed approaches using multi-attack datasets is important to show the real progress in the research area of robust NIDSs. The UPAS attack is stoppable using a simple adversarial training step that is performed without using a GAN. This demonstrates that approaches based on adversarial training can outperform those based on denoising autoencoders, and researchers should prioritise adversarial training when developing robust NIDSs. Moreover, perturbations generated using GANs can be replaced with simple handcrafted perturbations, such as splitting each malicious sample into three samples with a smaller number of packets.

6.4 Summary

In this chapter, we proposed an adversarial attack called UPAS to evaluate the robustness of NIDSs that are resilient to adversarial input perturbations. UPAS is a black-box attack because it requires no knowledge of the packets' payloads, the detection ML model or the training set of the ML model. UPAS was implemented on a multi-attack dataset and used to target two state-of-the-art NIDSs based on a LUCID model trained with GADoT and a RePO+ flow-based model. The results show that both NIDSs are vulnerable to the attack, thus, lack the expected robustness. To defend against the attack, we create an augmented training dataset such that each malicious sample in the CICIDS2017 training set is split into three samples with a smaller number of packets. After training with the augmented

dataset, a DL model becomes robust against UPAS, demonstrating that adversarial training can lead to robust NIDSs if the adversarial training dataset captures as many realistic perturbations as possible.

Chapter 7

Conclusion

The well-being and prosperity of our societies rely on the proper operation of several CIs, such as water treatment plants, power grids and telecommunication networks, the disruption of which may result in widespread harm. For this reason, such CIs have become targets for the proliferation of cyberattacks. Defending against such attacks often depends on ML-based ADSs, which establish a model of the normal behaviour of a CI using ML models to classify deviations that go well beyond the normality as anomalies. In this thesis, the focus was on the robustness of ML-based ADSs to adversarial and non-adversarial input perturbations.

The sources of non-adversarial input perturbations, such as normal behaviour evolution and noisy data readings, were identified and discussed. We provided examples of behaviour evolution in sensors and showed that this evolution can interfere with the correct functioning of ADSs. Noisy data readings have been identified as another source of natural perturbations that increase the number of false alarms and may allow attackers to hide their malicious activities. To address this problem, we proposed two frameworks, namely, DAICS and SiFA, for adaptive and robust anomaly detection in ICSs. DAICS is based on a wide and deep neural network and has a modular architecture to fit large ICSs. SiFA collects a buffer of historical

false alarms and suppresses every new alarm that is similar to these false alarms. It is worth noting that the SiFA is an extension of DAICS and might not be required in every environment. For example, when DAICS is so good that there are no false alarms to fill the SiFA's buffer.

The DAICS and SiFA were implemented and evaluated on a machine accelerated with a GPU, which might be unavailable in ICS environments, posing a challenge to deployment in real-world. Consequently, we carried out another experiment to evaluate the performance without GPU acceleration using a Colab notebook [120]. The results show that both frameworks are fast and that the tuning process takes less than 1.5 s for a batch of 32 samples. Moreover, we highlight that both frameworks do not interfere with the monitored system, making them ideal for legacy systems that can crash easily under active security measures.

To deploy DAICS and SiFA in a new environment, we need to collect a training dataset with a full operation cycle of the ICS. We also need to train the wide and deep neural network from scratch and tune the hyperparameters. This effort must be made at the beginning of every new deployment. To alleviate such a burden, we can apply transfer learning techniques to transfer the knowledge between different but related ICS infrastructure [121]. As a result, we avoid training a new model from scratch for every deployment as a single trained model can be deployed to several ICSs, and only a fine-tuning process is required to adapt the model to the new environment. However, there is no guarantee that the transferred model will achieve good results as such an achievement depends on the relevance between source and target environments [121]. Therefore, an interesting area for future work is to investigate the possibility of applying transfer learning techniques to the field of ICS anomaly detection.

Another issue is what happens if the ICS infrastructure is updated. If the update process includes adding/removing sensors and actuators, the

architecture of the wide and deep neural network will change, and thus, full retraining is required. On the other hand, if the update process affects only the behaviour of the industrial process, without adding/removing sensors and actuators, the updated infrastructure can cause DAICS to generate false positives. As mentioned in Chapters 3 and 4, false positives due to normal behaviour evolution can be mitigated by tuning the neural network according to the new behaviour.

Given that DAICS aims to detect anomalies in sensor readings, it can be used for predictive maintenance. Predictive maintenance employs machine learning techniques to learn the normal operation patterns of equipment and classifies a deviation from such patterns as a performance degradation that requires maintenance [122]. It aims to predict equipment faults before they occur and thus helps to avoid equipment failures and shutdowns in production processes. The process of detecting deviations from normal operation is similar to what DAICS does when detecting anomalies in ICS data. Therefore, a direction for future work is to evaluate DAICS on predictive maintenance datasets to determine its suitability for this use case.

The experiments to evaluate the DAICS and SiFA approaches were performed by feeding the dataset records into the wide and deep neural network and classifying each record as either normal or attack. In another experiment, we fed the dataset records into the neural network two consecutive times. The results show that the number of undetected attacks increases in the second round and that the neural network forgets the previously learnt normality, a phenomenon known as catastrophic forgetting. Catastrophic forgetting directly results from the so-called stability-plasticity dilemma in neural networks. Plasticity refers to the ability to learn new behaviours, and stability to the ability to maintain previous knowledge [123]. Therefore, an area for future work is to develop methods to overcome catastrophic

forgetting in the DAICS and SiFA frameworks.

Another area for future work is to localise the anomaly source in an ICS, which is essential for reactive attack mitigation and post-attack procedures. The existing approaches for localisation depend on ADS to identify devices that do not function as expected. However, a device can also behave abnormally if an attack targets an adjacent device. Therefore, further analysis is required to localise the compromised devices in a group of devices operating abnormally, i.e., to distinguish the compromised devices from those experiencing the consequences of an attack.

We showed that adversarial perturbations could be added to malicious network traffic to force the detection ML model to classify such traffic as benign. To address this problem, we introduced GADoT, an approach for adversarial training of ML-based attack detection systems. The results demonstrate that adversarial training using GADoT renders DDoS detection models robust to adversarial perturbations. GADoT can also be used to train multi-class classification models aiming at multi-attack detection (beyond DDoS). However, a preliminary study showed that the models remain vulnerable after adversarial training. Therefore, an interesting direction for future work is to adapt GADoT for the multi-class classification problem.

The evaluation of adversarial robustness is often error-prone, leading to robustness overestimation [119]. To confirm this observation, we proposed the UPAS attack, a universal adversarial attack against robust NIDSs. The experimental results demonstrate that state-of-the-art robust NIDSs are vulnerable to the attack and, thus, lack the expected robustness. The results of GADoT and UPAS demonstrate that adversarial training yields models with stronger robustness and highlight the need for a standardised evaluation of adversarial robustness in the problem space.

The arms race between creators of adversarial perturbations and defend-

ers is not going to stop in the near future. Attackers can always adapt their adversarial perturbations to evade the most recent defences. As a result, defenders should work on forecasting how adversarial attacks will evolve and what information is required to stop such attacks [124].

The experimental results obtained using DAICS and GADoT suggest that both solutions can be combined to form an NIDS that is robust to adversarial and non-adversarial input perturbations. In such a NIDS, the wide and deep neural network in DAICS is preceded by a feature extractor that processes the raw network traffic to extract data samples suitable for input to the neural network. The neural network must also be adapted to suit supervised learning, which is used to train DAICS to distinguish between benign and malicious network flows. DAICS training will be performed using a training dataset containing benign samples, malicious samples, and adversarial samples generated using GADoT. As in the original DAICS, the system operator would signal false alarms to the *few-time-steps* algorithm to learn from the users' evolving behaviour. The combination of DAICS and GADoT is expected to produce an attack detection system robust to perturbations regardless of the perturbation source.

Bibliography

[1] NCCIC/ICS-CERT. Cyber-attack against ukrainian critical infrastructure. `https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01`, access: 2019-07-25.

[2] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[3] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1093–1110, 2017.

[4] Eric D Knapp and Joel Thomas Langill. *Industrial Network Security: Securing critical infrastructure networks for smart grid, SCADA, and other Industrial Control Systems*. Syngress, 2014.

[5] Slashdot. Medical equipment crashes during heart procedure because of antivirus scan, 2016. `http://slashdot.org/story/310823`, access: 30-09-2019.

[6] Alvaro Cardenas and Bruno Crispo. Cyber-physical security and privacy [guest editors' introduction]. *IEEE internet computing*, 20(5):6–8, 2016.

[7] Tsui-Wei Weng. Evaluating robustness of neural networks, 2020. `https://hdl.handle.net/1721.1/129313`, access: 20-02-2022.

[8] James Aiken and Sandra Scott-Hayward. Investigating adversarial attacks against network intrusion detection systems in sdns. In *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–7. IEEE, 2019.

[9] Amir Mahdi Sadeghzadeh, Saeed Shiravi, and Rasool Jalili. Adversarial network traffic: Towards evaluating the robustness of deep-learning-based network traffic classification. *IEEE Transactions on Network and Service Management*, 18(2):1962–1976, 2021.

[10] Aditya Kuppa, Slawomir Grzonkowski, Muhammad Rizwan Asghar, and Nhien-An Le-Khac. Black box attacks on deep anomaly detectors. In *14th International Conference on Availability, Reliability and Security*, pages 1–10. ACM, 2019.

[11] Adel Bibi, Modar Alfadly, and Bernard Ghanem. Analytic expressions for probabilistic moments of pl-dnn with gaussian input. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9099–9107, 2018.

[12] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Robustness of classifiers: from adversarial to random noise. *Advances in Neural Information Processing Systems*, 29, 2016.

[13] Hossein Hosseini, Baicen Xiao, and Radha Poovendran. Google's cloud vision api is not robust to noise. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 101–105. IEEE, 2017.

[14] Maged Abdelaty, Roberto Doriguzzi-Corin, and Domenico Siracusa. Aads: A noise-robust anomaly detection framework for industrial control systems. In *Information and Communications Security: 21st International Conference, ICICS 2019, Beijing, China, December 15–17, 2019, Revised Selected Papers*, volume 11999, page 53. Springer Nature, 2020.

[15] Maged Fathy Abdelaty, Roberto Doriguzzi Corin, and Domenico Siracusa. Daics: A deep learning solution for anomaly detection in industrial control systems. *IEEE Transactions on Emerging Topics in Computing*, 2021.

[16] Maged Abdelaty, Sandra Scott-Hayward, Roberto Doriguzzi-Corin, and Domenico Siracusa. Gadot: Gan-based adversarial training for robust ddos attack detection. In *2021 IEEE Conference on Communications and Network Security (CNS)*, pages 119–127, 2021.

[17] MITRE. Adversarial ml threat matrix. `https://github.com/mitre/advmlthreatmatrix`, access: 2020-10-27.

[18] Keith Stouffer and Joe Falco. *Guide to supervisory control and data acquisition (SCADA) and industrial control systems security.* National institute of standards and technology, 2006.

[19] Peng Zhang. *Industrial control technology: a handbook for engineers and researchers.* William Andrew, 2008.

[20] Stamatis Karnouskos. Stuxnet worm impact on industrial cyber-physical system security. In *IECON 2011-37th Annual Conference of the IEEE Industrial Electronics Society*, pages 4490–4494. IEEE, 2011.

[21] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. A Dataset to Support Research in the Design of Secure Water Treatment Systems. Number October, pages 88–99. 2017.

[22] iTrust. iTrust Datasets. `https://itrust.sutd.edu.sg/testbeds/secure-water-treatment-swat`, access: 2019-01-25.

[23] Sridhar Adepu and Aditya Mathur. Generalized attacker and attack models for cyber physical systems. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 283–292. IEEE, 2016.

[24] Sridhar Adepu and Aditya Mathur. An investigation into the response of a water treatment system to cyber attacks. In *Proc. of IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*, 2016.

[25] Chuadhry Mujeeb Ahmed, Venkata Reddy Palleti, and Aditya P Mathur. Wadi: a water distribution testbed for research in the design of secure cyber physical systems. In *Proc. of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks*, 2017.

[26] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[27] Chuadhry Mujeeb Ahmed, Jianying Zhou, and Aditya P Mathur. Noise matters: Using sensor and process noise fingerprint to detect stealthy cyber attacks and authenticate sensors in cps. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 566–581. ACM, 2018.

[28] Chuadhry Mujeeb Ahmed, Martin Ochoa, Jianying Zhou, Aditya P Mathur, Rizwan Qadeer, Carlos Murguia, and Justin Ruths.

Noiseprint: attack detection using sensor and process noise fingerprint in cyber physical systems. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 483–497. ACM, 2018.

[29] Defu Wang, Xiaojuan Wang, Yong Zhang, and Lei Jin. Detection of power grid disturbances and cyber-attacks based on machine learning. *Journal of Information Security and Applications*, 46:42–52, 2019.

[30] Rohan Doshi, Noah Apthorpe, and Nick Feamster. Machine learning ddos detection for consumer internet of things devices. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 29–35. IEEE, 2018.

[31] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[33] Guilherme Serpa Sestito, Afonso Celso Turcato, Andre Luis Dias, Murilo Silveira Rocha, Maira Martins Da Silva, Paolo Ferrari, and Dennis Brandao. A method for anomalies detection in real-time ethernet data traffic applied to PROFINET. *IEEE Transactions on Industrial Informatics*, 14(5):2171–2180, 2018.

[34] Dmitry Shalyga, Pavel Filonov, and Andrey Lavrentyev. Anomaly detection for water treatment system based on neural network with automatic architecture optimization. *arXiv preprint arXiv:1807.07282*, 2018.

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[36] Zibin Zheng, Yatao Yang, Xiangdong Niu, Hong-Ning Dai, and Yuren Zhou. Wide and deep convolutional neural networks for electricity-theft detection to secure smart grids. *IEEE Transactions on Industrial Informatics*, 14(4):1606–1615, 2017.

[37] Moshe Kravchik and Asaf Shabtai. Detecting cyber attacks in industrial control systems using convolutional neural networks. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 72–83, 2018.

[38] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10. ACM, 2016.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[40] Jonathan Goh, Sridhar Adepu, Marcus Tan, and Zi Shan Lee. Anomaly detection in cyber physical systems using recurrent neural networks. In *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, pages 140–145. IEEE, 2017.

[41] Muna AL-Hawawreh, Nour Moustafa, and Elena Sitnikova. Identification of malicious activities in industrial internet of things based on deep learning models. *Journal of Information Security and Applications*, 41:1–11, aug 2018.

[42] Moshe Kravchik and Asaf Shabtai. Efficient cyber attacks detection in industrial control systems using lightweight neural networks. *arXiv preprint arXiv:1907.01216*, 2019.

[43] Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks. In *International Conference on Artificial Neural Networks*, pages 703–716. Springer, 2019.

[44] Ahmed A Abokifa, Kelsey Haddad, Cynthia Lo, and Pratim Biswas. Real-time identification of cyber-physical attacks on water distribution systems via machine learning–based anomaly detection techniques. *Journal of Water Resources Planning and Management*, 145(1):04018089, 2018.

[45] Hamed Farsi, Ali Fanian, and Zahra Taghiyarrenani. A novel online state-based anomaly detection system for process control networks. *International Journal of Critical Infrastructure Protection*, 27:100323, 2019.

[46] Qin Lin, Sridha Adepu, Sicco Verwer, and Aditya Mathur. Tabor: A graphical model-based approach for anomaly detection in industrial control systems. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 525–536. ACM, 2018.

[47] Luis Rosa, Tiago Cruz, Miguel Borges de Freitas, Pedro Quitério, João Henriques, Filipe Caldeira, Edmundo Monteiro, and Paulo Simões. Intrusion and anomaly detection for the next-generation of industrial automation and control systems. *Future Generation Computer Systems*, 119:50–67, 2021.

[48] Mauro Conti, Denis Donadel, and Federico Turrin. A survey on industrial control system testbeds and datasets for security research. *IEEE Communications Surveys Tutorials*, 23(4):2248–2294, 2021.

[49] Dmitry Shalyga, Pavel Filonov, and Andrey Lavrentyev. Anomaly detection for water treatment system based on neural network with automatic architecture optimization. *arXiv preprint arXiv:1807.07282*, 2018.

[50] Muhammad Qasim Ali, Ehab Al-Shaer, Hassan Khan, and Syed Ali Khayam. Automated anomaly detector adaptation using adaptive threshold tuning. *ACM Transactions on Information and System Security*, 15(4):1–30, 2013.

[51] Daehyung Park, Yuuna Hoshi, and Charles C Kemp. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3:1544–1551, 2018.

[52] Chuadhry Mujeeb Ahmed, Gauthama Raman MR, and Aditya P Mathur. Challenges in machine learning based approaches for real-time anomaly detection in industrial control systems. In *Proceedings of the 6th ACM on Cyber-Physical System Security Workshop*, pages 23–29, 2020.

[53] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*, pages 582–597. IEEE, 2016.

[54] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*, 2016.

[55] Fortinet. Types of cyber attacks, 2022. https://www.fortinet.com/resources/cyberglossary/types-of-cyber-attacks, access: 20-02-2022.

[56] CIS Center for Internet Security. Guide to ddos attacks. https://www.cisecurity.org/wp-content/uploads/2017/03/Guide-to-DDoS-Attacks-November-2017.pdf, access: 2021-07-25.

[57] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE communications surveys & tutorials*, 15(4):2046–2069, 2013.

[58] Cisco. Cisco annual internet report (2018–2023). https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html, access: 2020-08-05.

[59] A10. Aws hit by largest reported ddos attack of 2.3 tbps. https://www.a10networks.com/blog/aws-hit-by-largest-reported-ddos-attack-of-2-3-tbps/, access: 2020-08-05.

[60] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*, 2018.

[61] Roberto Doriguzzi-Corin, Stuart Millar, Sandra Scott-Hayward, Jesus Martinez-del Rincon, and Domenico Siracusa. Lucid: A practical, lightweight deep learning solution for ddos attack detection. *IEEE Transactions on Network and Service Management*, 2020.

[62] Mahmoud Said Elsayed, Nhien-An Le-Khac, Soumyabrata Dev, and Anca Delia Jurcut. Ddosnet: A deep-learning model for detecting network attacks. In *2020 IEEE 21st International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 391–396. IEEE, 2020.

[63] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.

[64] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of tor traffic using time based features. In *ICISSp*, pages 253–262, 2017.

[65] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. Developing realistic distributed denial of service (ddos) attack dataset and taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8. IEEE, 2019.

[66] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[67] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1332–1349. IEEE, 2020.

[68] Kaichen Yang, Jianqing Liu, Chi Zhang, and Yuguang Fang. Adversarial examples against the deep learning based network intrusion detection systems. In *2018 IEEE Military Communications Conference (MILCOM)*, pages 559–564. IEEE, 2018.

[69] Zheng Wang. Deep learning-based intrusion detection with adversaries. *IEEE Access*, 6:38367–38384, 2018.

[70] Olakunle Ibitoye, Omair Shafiq, and Ashraf Matrawy. Analyzing adversarial attacks against deep learning for intrusion detection in iot networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[71] Qiao Yan, Mingde Wang, Wenyao Huang, Xupeng Luo, and F Richard Yu. Automatically synthesizing dos attack traces using generative adversarial networks. *International Journal of Machine Learning and Cybernetics*, 10(12):3387–3396, 2019.

[72] Mohammad J Hashemi, Greg Cusack, and Eric Keller. Towards evaluation of nidss in adversarial setting. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks*, pages 14–21, 2019.

[73] Mohammad J Hashemi and Eric Keller. Enhancing robustness against adversarial examples in network intrusion detection systems. *arXiv preprint arXiv:2008.03677*, 2020.

[74] Rana Abou Khamis and Ashraf Matrawy. Evaluation of adversarial training on different types of neural networks in deep learning-based idss. *arXiv preprint arXiv:2007.04472*, 2020.

[75] Rana Abou Khamis, M Omair Shafiq, and Ashraf Matrawy. Investigating resistance of deep learning-based ids against adversaries using min-max optimization. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2020.

[76] Rui Shu, Tianpei Xia, Laurie Williams, and Tim Menzies. Omni: automated ensemble with unexpected models against adversarial evasion attack. *Empirical Software Engineering*, 27(1):1–32, 2022.

[77] Chaoyun Zhang, Xavier Costa-Pérez, and Paul Patras. Tiki-taka: Attacking and defending deep learning-based intrusion detection systems. In *2020 Cloud Computing Security Workshop (CCSW'20)*. ACM, 2020.

[78] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[79] Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O'Reilly. Adversarial deep learning for robust detection of binary encoded malware. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 76–82. IEEE, 2018.

[80] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[81] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. IEEE, 2009.

[82] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[83] Muhammad Usama, Muhammad Asim, Siddique Latif, Junaid Qadir, et al. Generative adversarial networks for launching and thwarting adversarial attacks on network intrusion detection systems. In *15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 78–83. IEEE, 2019.

[84] Imperva. TCP SYN Flood. `https://www.imperva.com/learn/ddos/syn-flood/`, access: 2020-30-11.

[85] Imperva. HTTP Flood. `https://www.imperva.com/learn/ddos/http-flood/`, access: 2020-30-11.

[86] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A Closer Look at Few-shot Classification. *arXiv preprint arXiv:1904.04232*, 2019.

[87] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[88] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

[89] P. Mulinka and P. Casas. Adaptive network security through stream machine learning. In *Proc. of ACM SIGCOMM*, 2018.

[90] P. Mulinka and P. Casas. Stream-based machine learning for network security and anomaly detection. In *Proc. of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, 2018.

[91] Moshe Kravchik, Battista Biggio, and Asaf Shabtai. Poisoning attacks on cyber attack detectors for industrial control systems. *arXiv preprint arXiv:2012.15740*, 2020.

[92] Alessandro Erba, Riccardo Taormina, Stefano Galelli, Marcello Pogliani, Michele Carminati, Stefano Zanero, and Nils Ole Tippenhauer. Constrained concealment attacks against reconstruction-based anomaly detectors in industrial control systems. In *Annual Computer Security Applications Conference*, pages 480–495, 2020.

[93] Luis Garcia, Ferdinand Brasser, Mehmet Hazar Cintuglu, Ahmad-Reza Sadeghi, Osama A Mohammed, and Saman A Zonouz. Hey, my malware knows physics! attacking plcs with physical model aware rootkit. In *NDSS*, 2017.

[94] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[95] Beckhoff Information System. Continuous Valve Actuator. `https://infosys.beckhoff.com/english.php?content=../content/1033/tcbaframework/html/TcBAManager_ValveActuatorType03.htm`, access: 2021-01-01.

[96] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[97] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 2017.

[98] ODVA. Technology overview series: Ethernet/ip. Technical report, 2016.

[99] Brendan Galloway and Gerhard P Hancke. Introduction to industrial control networks. *IEEE Communications surveys & tutorials*, 15(2):860–880, 2012.

[100] Graham C Goodwin, Daniel E Quevedo, and Eduardo I Silva. Architectures and coder design for networked control systems. *Automatica*, 44(1):248–257, 2008.

[101] Xi-Sheng Zhan, Zhi-Hong Guan, Fu-Shun Yuan, and Xian-He Zhang. Performance analysis of networked control systems with snr constraint. *International Journal of Innovative Computing, Information and Control*, 8(12):8287–8298, 2012.

[102] Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks. In *International Conference on Artificial Neural Networks*, pages 703–716. Springer, 2019.

[103] Hongda Tian, Nguyen Lu Dang Khoa, Ali Anaissi, Yang Wang, and Fang Chen. Concept drift adaption for online anomaly detection in structural health monitoring. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2813–2821, 2019.

[104] Xiangyu Li, Kun Cheng, Tao Huang, and Sichao Tan. Research on false alarm detection algorithm of nuclear power system based on bert-sae-iforest combined algorithm. *Annals of Nuclear Energy*, 170:108985, 2022.

[105] Shah Ahsanul Haque and Syed Mahfuzul Aziz. False alarm detection in cyber-physical systems for healthcare applications. *Aasri Procedia*, 5:54–61, 2013.

[106] Mohammed GH Al Zamil, Samer Samarah, Majdi Rawashdeh, M Shamim Hossain, Mohammed F Alhamid, Mohsen Guizani, and Awny Alnusair. False-alarm detection in the fog-based internet of connected vehicles. *IEEE Transactions on Vehicular Technology*, 68(7):7035–7044, 2019.

[107] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[108] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[109] Giovanni Apruzzese, Mauro Andreolini, Mirco Marchetti, Vincenzo Giuseppe Colacino, and Giacomo Russo. Appcon: Mitigating evasion attacks to ml cyber detectors. *Symmetry*, 12(4):653, 2020.

[110] Roberto Doriguzzi-Corin. LUCID source code. `https://github.com/doriguzzi/lucid-ddos`, access: 2020-30-11.

[111] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[112] Generative adversarial networks. `https://developers.google.com/machine-learning/gan/generator`, access: 2020-11-25.

[113] Salvatore Sanfilippo, N Jombart, D Ducamp, Y Berthier, and S Aubert. Hping. `www.hping.org`, access: 2020-08-05.

[114] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security*, 31(3), 2012.

[115] Philippe Biondi. Scapy: explore the net with new eyes. *Technical report, EADS Corporate Research Center*, 2005.

[116] Loic. `https://www.imperva.com/learn/ddos/low-orbit-ion-cannon`, access: 2020-08-05.

[117] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. `http://tensorflow.org/`, access: 2020-12-04.

[118] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification. *arXiv preprint arXiv:1809.08037*, 2018.

[119] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670*, 2020.

[120] Colab. `https://colab.research.google.com/`, access: 2022-07-05.

[121] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[122] Thyago P Carvalho, Fabrízzio AAMN Soares, Roberto Vita, Roberto da P Francisco, João P Basto, and Symone GS Alcalá. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137:106024, 2019.

[123] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks.

IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.

[124] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. Arms race in adversarial malware detection: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–35, 2021.