University of Salerno

Department of Computer Science

Dottorato di Ricerca in Informatica
Curriculum Computer Science and Information Technology
XXXV Ciclo

Tesi di Dottorato / Ph.D. Thesis

A Study of some ML and DL-based Strategies for Network Security

Eslam FARSIMADAN

Supervisor: **Prof. Francesco PALMIERI**

PhD Program Director: **Prof. Andrea DE LUCIA**

A.A 2022/2023

Family means nobody gets left behind or forgotten.

Dedicated to my family.
For their endless love, support, and encouragement.

No one who achieves success
does so without acknowledging the help of others.
The wise and confident
acknowledge this help with gratitude.

— Alfred North Whitehead

ACKNOWLEDGMENTS

Dear Prof. Francesco Palmieri, words cannot express my gratitude for the guidance, support, and mentorship you have provided me throughout my Ph.D. journey. Your expertise, insight, and unwavering commitment have been a true inspiration, and I am deeply grateful for your impact on my life and research.

Your dedication to your students is unparalleled, and your commitment to excellence is an example to us all. I am truly honored to have had the opportunity to work with you and to benefit from your wisdom, guidance, and encouragement. Your insightful and constructive feedback has been instrumental in shaping my research and bringing it to completion, and I am proud to have learned from the best.

I would also like to extend my sincere thanks to the reviewers who have generously given their time and expertise to evaluate my work. Your suggestions and insights have been invaluable in improving the quality of my research, and I am genuinely grateful for the opportunity to incorporate your feedback into my thesis.

Finally, I would like to acknowledge the support of my colleagues, friends, and family (especially my beloved wife, Leila), who have been a constant source of encouragement and inspiration throughout this journey.

Thank you all for being a part of my life and helping me reach this fantastic milestone. I am eternally grateful for your support and guidance, and I will cherish these memories.

With heartfelt appreciation,
Eslam Farsimadan

ABSTRACT

The Internet and advanced communication networks, such as IoT and cellular networks, produce enormous and diverse traffic data flows. The behavior of network traffic in these networks is highly intricate due to factors like device mobility and network heterogeneity. As a result, conventional network security and management methods struggle to handle the challenges of securing, monitoring, and analyzing the network and data. These challenges include issues such as the efficacy of classification and detection strategies, precision, accuracy, and the ability to process big data in real-time.

Recently, machine learning and deep learning have proven to be highly effective in addressing network security concerns and have demonstrated superiority over traditional methods. Consequently, researchers in the field of networking are turning to these machine learning and deep learning models for network security and management.

Motivated by the success and effectiveness of these models, this thesis concentrates on addressing two crucial and challenging issues in network security through dynamic analysis, machine learning, and deep learning models, with a particular emphasis on artificial neural networks. More precisely, it presents new learning-based techniques for attack classification and malware detection.

First, a general introduction with some motivations for this thesis is presented. Then, the state-of-the-art is investigated, and the most effective artificial intelligence-based methods related to the above-mentioned network security aspects are reported. After that, the primary materials and preliminaries for constructing the proposed models, such as neural network types, recurrence plots, and so on, are introduced in detail. Finally, the proposed methods for attack classification and malware detection are investigated in terms of mathematical background, model architecture, experimental setting, and evaluation.

Moreover, the proposed methods are analyzed from a theoretical perspective and through specific performance evaluation experiments on real network traffic datasets. The obtained results, in the presence of several unbalanced datasets instances, prove the effectiveness of the proposed approaches.

Keywords: Network Security, Attack Detection, Attack Classification, Malware Detection, Machine Learning, Deep Learning, Neural Networks, Recurrence Plots, Markov Chain

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

IoT	Internet of Things
AI	Artificial Intelligence
FL	Federated Learning
IDS	Intrusion Detection Systems
MAC	Media Access Control
non-IID	non-Independent and Identically Distributed
ML	Machine Learning
SL	Supervised Learning
K-NN	K-nearest neighbors
SVM	Support Vector Machines
UL	Unsupervised Learning
PCA	Principal Component Analysis
SSL	Semi-Supervised Learning
RL	Reinforcement Learning
SARSA	State-Action-Reward-State-Action
DQN	Deep Q network
DDPG	Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimization
PPO	Proximal Policy Optimization
DL	Deep Learning
ANN	Artificial Neural Network
NN	Neural Network
DNN	Deep Neural Network
DAG	Directed Acyclic Graph
CNN	Convolutional Neural Network
ReLU	Rectified Linear activation fUnction
RNN	Recurrent Neural Network

BPTT Backpropagation Through Time

LSTM Long Short-Term Memory

RTRL Real-Time Recurrent Learning

AEs Autoencoders

PCA Principal Component Analysis

LDA Linear Discriminant Analysis

DFA Discriminant Function Analysis

t-SNE t-distributed Stochastic Neighbor Embedding

SAE Sparse AE

SNN Stacked Neural Network

RPs Recurrence plots

AMI Average Mutual Information

FNN False Nearest Neighbours

API Application Programming Interface

1

INTRODUCTION

The Internet is one of the most important inventions of the 21st century that has dramatically impacted our daily lives. It has transcended borders and revolutionized the way we communicate, play games, work, shop, socialize, listen to music, watch movies, order food, pay bills, and more. As its name suggests, the Internet is a network of networks comprising various small, medium, and large networks and has become a fundamental part of our lives. Many people in today's generation depend on the Internet for their professional, social, and personal needs and activities (Figure 1.1) [1]. The rise of the Internet, the Internet of Things (IoT), and advanced networking technologies have made the world increasingly interconnected. Consequently, vast amounts of personal, commercial, military, and government information are stored on networking infrastructures around the world. These networks and advanced technologies are crucial for the functioning of the Internet and various related applications in fields such as engineering, computer science, and beyond.

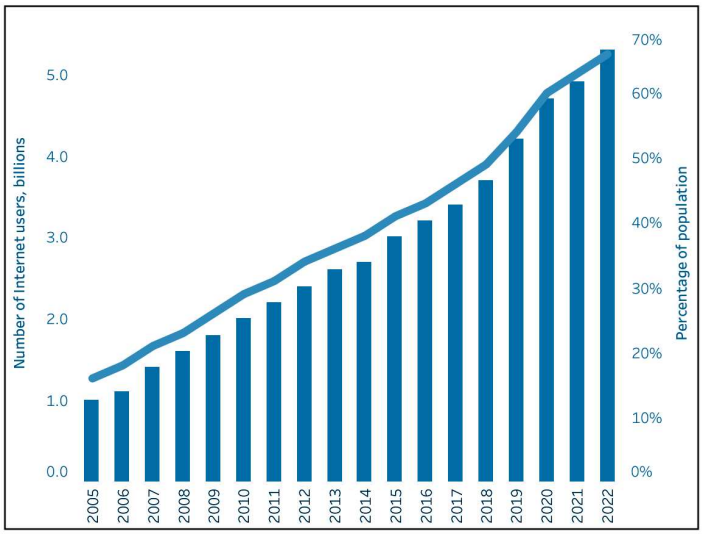

Figure 1.1: Individuals using the Internet [1].

One of the critical and most complex challenges in these interconnected networks and advanced networking technologies is ensuring the security of these networks and applications. Network security involves continuous efforts to evaluate the security status of the network, implement protective and preventative measures against security threats, and establish mechanisms for quickly detecting security issues. Moreover, it has some protocols, procedures, and techniques for responding to attacks. In simpler terms, network security is about protecting our networks and data from breaches, intrusions, and other security threats. It is a broad and encompassing term that encompasses hardware and software solutions, as well as policies, rules, and configurations related to network use, accessibility, and overall threat protection [2].

The report by Cybersecurity Ventures in [3] states that network security problems are becoming more prominent, sophisticated, and expensive. It is estimated that the cost of global network security and cybersecurity issues will increase by 15% each year over the next four years, reaching 10.5 trillion USD annually by 2025, up from 3 trillion USD in 2015. This is the largest transfer of wealth in history and poses a risk to innovation and investment incentives. It is also significantly larger than the yearly damage caused by natural disasters and more profitable than the combined global trade of all major illegal drugs. Additionally, according to the 2021 Cyberthreat Defense Report (CDR) by the CyberEdge Group, 86% of organizations experienced at least one successful cyber attack last year (Figure 1.2). Hence, it is of utmost importance to enhance the security of communication systems and networks against security issues and cyber threats, mainly as people are relying more on wireless networks such as cellular networks and WiFi for daily activities (e.g., online shopping, online banking, and internet-based business).

Even though network security is crucial for both the Internet and newly developed networks, there is a noticeable absence and significant lack of simple-to-apply and easy-to-implement security measures. Indeed, there exists a communication gap between the developers of network security technologies and developers of networks. Network design is a well-developed process. In contrast to network design, secure network design is

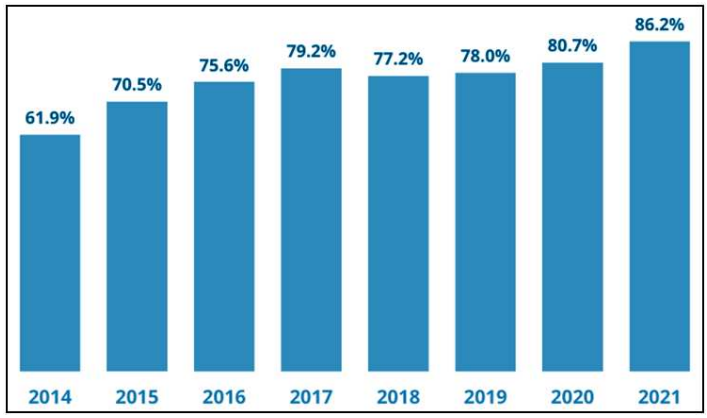

Figure 1.2: Percentage of organizations compromised by at least one successful attack [4].

not a well-developed process. When it comes to network security, it is crucial to remember that the entire network must be secure, not just specific components. Indeed, network security involves more than just ensuring the security of the computers at the start and end of the communication. The communication channel used to transmit data must also be secure and protected from potential security breaches.

Therefore, nowadays, network security should increasingly be gaining attention as the Internet and networking technologies expand. Initially, it was assumed that new security methods would be actively researched due to the significance of the field. However, current developments and research in network security are not very promising, impactful, or impressive. Unfortunately, a large portion of network security research is still focused on improving the same technologies currently being used that have been proven not to be effective enough.

Consequently, and due to the expanding traffic flow in advanced communication systems like cellular and IoT networks, network security-related issues such as detecting malware on IoT devices, detecting anomalies and attacks, and classifying network traffic have become crucial research areas in the field of network security. Recently, to tackle these challenges, new Artificial Intelligence (AI)-based classification and detection strategies, such as machine learning and artificial neural network techniques,

have been explored as effective methods for effectively identifying abnormal events within normal traffic and ensuring secure networks.

Machine learning (ML)-based models have been highly successful in addressing security issues in networking technologies. They can use large datasets to train the models in centralized and decentralized approaches. However, building a comprehensive knowledge base on emerging security threats is currently a slow, complex, and challenging process [5].

Artificial Neural Networks (ANNs) are a machine learning technique that takes inspiration from the functioning of neurons in the human brain. In other words, an ANN is an adaptive system that learns through interconnected nodes in a layered structure that mimics the structure of the human brain. Therefore, they are especially suitable for modeling non-linear relationships and are typically used to perform pattern recognition, classify objects, and control systems. ANNs rely on training data to learn and improve their accuracy over time, and once these learning algorithms are fine-tuned, they are strong tools in computer science, particularly network security and related classification tasks. Moreover, ANN-based approaches are a crucial technology causing innovation in many systems and tasks, including network traffic classification, detection tasks, and feature extraction. An ANN consists of an input layer, one or more hidden layers, and an output layer. In each layer, there are several nodes or neurons, and the nodes in each layer use the outputs of the nodes in the previous layer as inputs, such that all neurons interconnect through the different layers.

ANNs with multiple layers are referred to as Deep Learning (DL) or Deep Neural Networks (DNNs). DNNs have recently begun to outperform traditional ANN approaches in various domains, particularly security issues and pattern recognition. That is because of their ability to handle a massive amount of data. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Autoencoders (AEs), and their related combinations are eligible under the umbrella of DNNs, which can be employed to deal with security issues related to IoT domains and ICT infrastructures.

1.1 MOTIVATIONS

However, some considerations should be taken into account in the case of using ML, DL, and in general learning-based models. Building privacy-preserving models and choosing the appropriate form of datasets are among the challenging issues in these models.

Since the involved organizations (developers and/or end-users) with network security are often unwilling to share their own data since they are concerned about disclosing their intellectual property and sensitive data about their IoT applications and systems, building privacy-preserving models is a critical task in network security. Federated Learning (FL)-based approaches represent recent privacy-preserving solutions that employ the ML and DL models' capabilities in several classification and detection tasks without the need to share critical data [6]. In fact, FL is an ML setting where multiple entities collaborate in solving a learning problem without directly exchanging data. A central server coordinates the federated training process. FL allows multiple parties to jointly train a model on their combined data without having to compromise the privacy of any of the participants. Since private data never leaves the local devices, FL can provide strong privacy guarantees to the participants. These guarantees can be made rigorous by applying encryption techniques to the communicated parameter updates or concealing them with differentially private mechanisms.

On the other side, datasets and how to use them to feed ML, DL, and other learning-based models are crucial. In other words, these models (specifically for network security applications) should be fed with suitable datasets in good ways to be effective and robust. In this regard, some useful techniques and materials can be used to reconstruct and reshape datasets to feed the models effectively.

Recurrence Plots (RPs) are a helpful tool in visualizing the states of complex systems, where recurrence plays a significant role in the temporal evolution of the systems' dynamic trajectories [7]. Hence, by utilizing the time series of a single observable variable, a topologically equivalent representation of the behavior of the original multi-dimensional system can be reconstructed.

Consequently, since RPs can visualize complex systems' states by representing discriminative and dynamic characteristics, they might be employed to train a CNN capable of providing good classification performance without being adversely affected by obfuscation techniques.

Furthermore, Markov chains are one of the most effective cutting-edge dynamic analysis-based strategies that can model the API calls invoked by malware applications and construct the representative behavioral patterns of particular malware families [8]. More precisely, they consider the sequence of API calls to model the application-related behavior as a graph where each node represents a unique API, while each edge represents the transition probability between two APIs. Markov chain-based detectors have also been proven resistant to evasion efforts carried out by selectively inserting irrelevant API calls throughout malicious [9].

With these motivations, in this thesis, we focus on the effectiveness of some ML, DL, and dynamic analysis-based models in dealing with crucial issues in network security and the related most disparate classification tasks (e.g., network traffic classification, encrypted traffic, attack detection/classification, and malware detection). The main contribution of this thesis is based on some of our papers that are [10–12].

In the following, we briefly explain some of the important network security issues that are investigated in this thesis, our motivations, and our contribution. In the next chapters, we will focus on these issues in detail.

1.2 ATTACK DETECTION/CLASSIFICATION

Attack detection and classification are critical data analysis processes for spotting network intrusions and attacks. Detecting and classifying network attacks is the challenge of monitoring and distinguishing anomalous traffic patterns, flows, and activities from the network's regular anticipated behavior, which might compromise information system security. Attack detection, classification, and prevention are at the forefront of the information security landscape as organizations and governments look for trustworthy solutions to safeguard their information assets from

unwanted disclosures and unauthorized access. Furthermore, as demand and applications in broad fields such as security, health and medical risk, financial surveillance, risk management, and AI safety grow, attack detection/classification plays an increasingly significant role, as highlighted in numerous communities such as machine learning, computer vision, data mining, and statistics [13].

The goal of DL for attack detection and classification is to develop feature representations or attack scores using neural networks. Numerous deep attack detection/classification techniques have been developed, showing much superior performance than traditional strategies in tackling complex detection and classification issues in multiple real-world applications. DL methods, such as CNNs and AEs, provide end-to-end optimization of the whole attack detection pipeline, as well as the learning of representations especially tuned for attack detection and classification. These two characteristics are critical for addressing the primary attack detection/classification issues, but traditional techniques lack them. Regardless of the data type, they mainly assist in enhancing the usage of labeled normal data or some labeled attack data, hence decreasing the requirement for extensive labeled data as in fully-supervised scenarios [14].

With these motivations and the importance of network attack detection and classification in network security, this thesis introduces a network attack classification method based on a particular stacked neural network [11].

Indeed, we explore the theory of dynamic non-linear systems for effectively capturing and understanding the innermost and more expressive Internet traffic dynamics with the aim of reliably recognizing and classifying network attacks and anomalies. Consequently, a novel IoT-related network attacks classifier is proposed by combining the capabilities of AEs in automatically finding relevant features and the effectiveness of characteristics coming from non-linear analysis theory arranged as recurrence plots. More precisely, we consider statistical information related to benign and malicious traffic-related activities available in the CIC-IDS2017 dataset as raw input data. Then, we employ the corresponding multi-channel image representation of recurrence plots and a Convolutional Sparse Autoencoder (CNN-SAE) to per-

form automatic interpretation of recurrence plot images, intending to use the reliable discriminating power of such non-linear structures in attack classification tasks. The achieved results show the effectiveness of the proposed classifier also in the presence of several unbalanced datasets partitions.

1.3 MALWARE DETECTION

Malware, which stands for malicious software, is designed to access or be installed on computers, cellphones, and other similar devices without the user's consent or permission. For the profit of a third party, they carry out undesired tasks on the host computer, smartphone, and other devices. Malware has the potential to significantly reduce the host machine's performance. Malware comes in a wide variety, from simple programs meant to annoy users to sophisticated programs that steal critical information from the host system and transfer it to remote servers. Furthermore, there are several varieties of malware on the Internet. Adware, Browser Hijacking Software, Spyware, Viruses, Worms, Scareware, and Trojan Horses are some of the more well-known ones [15].

Many computer and IoT systems are susceptible to attacks and malware infection due to broad Internet use. Therefore, the early recognition of malware is necessary to secure the systems, data, and information. Several malware detection techniques have recently been presented by researchers [16]. However, due to obfuscation and evasion strategies, as well as the variety of malicious behavior brought on by the rapid rate at which new malware and malware variants are developed every day, multiple challenges prevent these solutions from properly identifying various forms of malware, particularly zero-day attacks [17].

With the extensive use of ML and DL in recent years, new methods for identifying malware have developed using these techniques, and malware detection models have become significantly more successful. As a result, considerable research has been done on the techniques of malware detection utilizing ML and DL to deal with the increasing proliferation of malware [18]. However, because these models are frequently trained on private application data, many of the participants are hesitant to con-

tribute their data for this purpose. Thus, some FL-based solutions that employ ML models for malware detection and classification without disclosing user data are becoming increasingly prevalent [19, 20].

Aimed by this motivation, in this thesis, a new ML-based malware detection model, which uses the benefits of federated Markov chains, for Android-based IoT devices is introduced [10].

We propose a federated Markov chains paradigm, which makes data owners proactive contributors to the ML-based detector model building, giving them the means to timely update a global model without sharing their private raw data. More precisely, Markov chains and associative rules are used within a federated logic, in which users independently process the raw data of each application and then send the extracted information to a central server, primarily dedicated to setting up and sharing the detector in a way that protects user privacy. We validate the effectiveness of the presented method by using a malware dataset composed of several famous malware families. Finally, we analyze the required temporal effort through a dedicated performance assessment, where several dataset partitions, splitting criteria, and clients have been considered. The related performances are comparable for the considered dataset in the presence of non-IID data.

Therefore, as mentioned before, the main objective of the thesis is to investigate the effectiveness of some dynamic analysis and AI-based models, specifically artificial neural networks, for network security (network attacks classification and malware detection on IoT devices).

1.4 OUTLINE OF THE THESIS

Based on the objectives, the thesis is organized as follows:

- Chapter 1 is for a general introduction. This chapter starts with a discussion of the current status of network security and introduces some of the most crucial issues, challenges, motivations, and countermeasures in this area. It also briefly discusses artificial intelligence as an effective solution to face network security problems. Moreover, it remarks on the motivations of this thesis, and some of the

basic information about the original contributions of this thesis, which are some aspects of network security such as attack detection/classification and malware detection, are presented. Finally, this chapter terminates by highlighting the outlines of this thesis.

- Chapter 2 reviews the recent studies on the network security aspects that are addressed in this thesis. In fact, this chapter presents the related works about the most relevant ML, DL, and dynamic analysis-based approaches applied in the before-mentioned classification and detection tasks.

- Chapter 3 presents some preliminaries. The approaches presented in this thesis make use of different ML and DL models. Moreover, different types of neural networks are employed to provide robust classifiers or detectors, which are suitable for most network security tasks. For the sake of clarity, in this chapter, the basic theoretical backgrounds of machine learning, deep learning, the employed neural networks, recurrence plots, and the Markov chain are briefly presented.

- Chapter 4 aims to propose a novel network attacks classifier. Accordingly, in this chapter, the capabilities of a convolutional autoencoder-based neural network are exploited for mining relevant features from traffic flows. Indeed, nonlinear characteristics, essentially represented as attractors' trajectories in phase space, arranged as recurrence plots, and the most relevant spatial features extracted from them by using a convolutional sparse autoencoder, are combined for supporting attack classification tasks. Moreover, a formal description of the overall workflow is provided to motivate how the proposed approach, from the theoretical point of view, improves and affects the feature extraction process, respectively.

- Chapter 5 presents a federated Markov chains-based model for malware detection in Android-based IoT scenarios. This paradigm makes data owners proactive contributors to the ML-based detector model building, giving them the means

to timely update a global model without sharing their private raw data. In other words, in this chapter, a federated architecture is presented to support the rules mining process as multiple Markov chains, and then, the resulting associative rules-based detector is exploited to recognize and classify several malware families by considering both centralized and decentralized data.

- Chapter 6 shows the conclusions of this thesis and future work.

2 STATE OF THE ART

This chapter investigates the recent studies on the network security aspects that are addressed in this thesis. Indeed, this chapter presents the related works about the most relevant ML and DL-based approaches applied in attack detection and classification and dynamic analysis-based methods for malware detection tasks. Therefore, it is organized into two sections, each one devoted to the state-of-the-art related to one of the above-mentioned issues.

2.1 ATTACK DETECTION AND CLASSIFICATION

DL is widely used for network security, particularly for network Intrusion Detection Systems (IDS). A study by Papamartzivanos et al. [21] introduced a DL-based, self-adaptive, and autonomous misuse detection system that utilized Autoencoder and sparse Autoencoder. It combined self-taught learning and MAPE-K frameworks to create a scalable and autonomous IDS with 77.99% accuracy, which was higher than the static technique's 59.71% accuracy. The complexity of modern communication systems and big traffic data make it difficult for traditional techniques and classical ML to handle the IDS task. Naseer et al. [13] studied the use of DL for anomaly detection systems and proposed various DL-based methods such as CNN, AE, and RNN, along with traditional ML models such as Support Vector Machine (SVM), Decision Tree, Nearest Neighbour, and Random Forest. They evaluated their methods using the NSLKDD dataset [22] and found that DL-based anomaly detection is suitable for real-world applications. The authors aimed to compare the effectiveness of DL models to shallow models in anomaly detection and found that DL models outperform shallow models in classification metrics like accuracy and precision, but take more time for training and testing. Jiang et al. [23] proposed using a CNN to detect virtual Media Access Control (MAC) spoofing attacks by gathering physical features from channel state information.

The authors in [24] focus on anomaly detection in cloud data center networks using the Gray Wolf Optimization (GWO) algorithm and CNN. They assert that their method is ideal for real-time anomaly detection of network log big data. They tested their approach using the DARPA'98, KDD'99, and synthetic datasets and found it to be superior to existing methods in the literature. The approach achieved an accuracy of 97.92% on the DARPA'98 dataset and 98.42% accuracy on the KDD'99 dataset. The authors highlight that current anomaly detection methods are inefficient and result in high computational complexity and false positives, especially in the context of real-time big data anomaly detection.

Yousefi-Azar et al. demonstrated the potential of using AE in cybersecurity, specifically for anomaly and malware detection [25]. The authors used a single AE model with the same architecture for both tasks and obtained favorable results due to AE being an unsupervised generative model that can learn the inherent representation of traffic data. The proposed AE had an accuracy of 83.34% and outperformed traditional ML models such as Decision Tree, Gaussian Naïve Bayes, and Fuzzy classifier. Malaiya et al. [26] studied the use of DL for anomaly detection in network traffic data. They evaluated different DL models such as fully connected networks, Sequence-to-Sequence, and Variational Autoencoder. They noted that the non-linear nature of network traffic data makes it difficult for classical ML techniques like SVM to perform well in anomaly detection.

The author in [27] proposed using Stacked Autoencoders (Stacked AEs) in 2017 for anomaly detection and attack categorization in IEEE 802.11 networks. The author first studied the threats and attacks in this network and identified the challenges in achieving high accuracy in attack classification. Then, a Stacked AE method was proposed for anomaly detection and classification. The method can automatically learn the critical features of the data and has an accuracy of 98.66%.

Chen et al. addressed security in mobile edge computing for transportation systems in [28]. The authors highlighted the growing communication volume among connected edge devices and the resulting security challenge. To overcome this, they proposed a deep belief network-based feature learning method to detect unknown attacks in mobile edge computing. Their proposed

method was compared to 4 classical ML algorithms (Softmax Regression, Decision Tree, SVM, and Random Forest) and showed improved accuracy.

Notwithstanding the significant effort made in annotating IoT traffic records, the number of labeled records is still limited, raising the difficulty in identifying attacks and intrusions. To address this issue, Abdel-Basset et al. [29] introduced a semi-supervised DL approach for intrusion detection called SS-Deep-ID, which consists of a multi-scale residual temporal convolutional (MS-Res) module that finetunes the network capability in learning spatio-temporal representations. The key in the MS-Res module is the dilated causal convolutions (DC-Conv) [30], and a traffic attention module is incorporated to help the network emphasize the most significant features for detecting intrusions.

Nie et al. [31] developed an IDS based on the Deep Deterministic Policy Gradient (DDPG) algorithm [32]. Their method first extracts the statistical features of prior network traffic to capture the trends of traffic flows and perform traffic prediction. Then, the developed traffic predictors are employed in combination with a suitable threshold to enable intrusion detection. The CIC-DDoS2019 dataset [33] was used to evaluate the proposed model. The achieved performance included 99% precision with a false positive rate of 1.21%, outperforming PCA and Sparse Regularized Matrix Factorization (SRMF) [34].

Wang et al. [35] proposed a federated anomaly detection system employing deep reinforcement learning to enable multiple parties to jointly learn an accurate deep model while preserving the data itself locally and confidential. Another significant advantage of using federated distribution instead of a centralized architecture is that unexpected intrusions in one or more client systems do not affect the whole system. However, since federated learning employs secure aggregation to protect the confidentiality of the local models, it cannot detect anomalies in the participants' contributions to the joint model.

The discussed ML and DL-based methods for network security are summarized in Table 2.1.

On the other side, Recurrence Plots (RPs) have been applied in many scientific fields to study and better understand complex systems' behavior, including astrophysics, earth sciences,

Author(s)	Method(s)	Advantages
Papamartzivanos et al. [21]	AEs, Sparse AEs, self-taught learning, MAPE-K	Scalable and autonomous IDS
Naseer et al. [13]	CNN, AE, RNN, SVM, Decision Tree, Nearest Neighbor, Random Forest	DL-based anomaly detection suitable for real-world applications
Jiang et al. [23]	CNN	Detection of virtual MAC spoofing attacks
Garg et al. [24]	GWO, CNN	Ideal for real-time anomaly detection of network log big data
Yousefi-Azar et al. [25]	AEs	Anomaly and malware detection model capable of learning the inherent representation of traffic data
Malaiya et al. [26]	Fully connected networks, LSTM Sequence-to-Sequence, Variational AEs	99% of binary classification accuracy for network anomaly detection on public data sets
Thing [27]	Stacked AEs	High accuracy in attack classification
Chen et al. [28]	Deep belief network	Improved accuracy in detecting unknown attacks in mobile edge computing
Abdel-Basset et al. [29]	DC-Conv, Attention module	Efficiency of intrusion detection and increasing the robustness of performance while maintaining computational efficiency
Nie et al. [31]	DDPG algorithm	99% precision with an false positive rate of 1.21%
Wang et al. [35]	Federated Deep Reinforcement Learning empowered Anomaly Detection (FLAD)	In order to prevent privacy leakage, abnormal actions of users are detected on time

Table 2.1: An overview of the discussed ML and DL-based network security works.

engineering, biology, cardiology, and neuroscience, as extensively surveyed in [36].

In particular, DL-based models have been widely employed for extracting non-linear features from RPs calculated on raw data

samples [37–39]. For instance, Kirichenko et al. [40] have investigated the effectiveness of RPs derived from electroencephalograms (EEG) time series concerning various human conditions. The obtained results, carried out using the Epileptic Seizure Recognition dataset [41], have proven the effectiveness of CNNs in detecting both seizure and no-seizure situations from the related plots with an average accuracy of 98.40%. Tziridis et al. [42] have deeply investigated the RPs related to EEG signals by applying to them several noise levels. More precisely, they have perpetuated the input signal by adding, each time, one noise level and then classifying the related RPs through several famous state-of-the-art CNNs, such as AlexNet [43], ResNet18 [44], DenseNet121 [45], and VGG16 [46], respectively. The obtained results have proven, for each considered dataset, the validity of RPs in the presence of several noise levels by achieving an average accuracy between 95% and 97%.

Furthermore, RPs have also been investigated in strictly-related Computer Science fields, like Malware classification. In this direction, Sartoli et al. [47] have employed a CNN to perform a malware classification task by considering the plots derived from the malware binary images. More precisely, the obtained results, carried out from the Microsoft Malware Classification Challenge dataset [48] have proven the effectiveness of the proposed approach with an average accuracy of 96.7% in the presence of nine unbalanced malware families.

Internet traffic analysis solutions have also been investigated using non-linear features arranged as RPs, which offered significant advantages in identifying more complex and unidentified traffic categories [7]. More precisely, since RPs can visualize complex systems' states by representing discriminative and dynamic characteristics [49, 50], they might be employed to train a neural network or machine learning mechanism capable of providing good classification performance without being adversely affected by obfuscation techniques. The first network anomaly detection experience leveraging non-linear methods, particularly Recurrence Quantification Analysis, has been presented in work [51]. Similar techniques have also been employed [52], relying on time-dependent Unthresholded Recurrence Plots for representing the traffic time series, with the goal of capturing their non-linear

features, then analyzed through an Extreme Learning Machine Autoencoder.

The discussed RP-based methods are summarized in Table 2.2.

Author(s)	Method(s)	Advantages
Kirichenko et al. [40]	RPs, CNN	Detecting both seizure and no-seizure situations from the related plots with an average accuracy of 98.40%
Tziridis et al. [42]	RPs, CNNs	An average accuracy between 95% and 97%
Sartoli et al. [47]	RPs, CNNs	An average accuracy of 96.7% in the presence of nine unbalanced malware families
Palmieri et al. [7]	RPs, Analysis of non-stationary hidden transition patterns of IP traffic flows	Effective for providing a deterministic interpretation of recurrence patterns in traffic flows
Palmieri et al.[51]	Recurrence Quantification Analysis, Discriminative Restricted Boltzmann Machine	Detecting any type of unknown anomalous events
Hu et al. [52]	Unthreholded RPs, Extreme Learning Machine AE (ELM-AE)	Performing well on complex nonlinear systems using unlabeled datasets

Table 2.2: An overview of the discussed RP-based methods.

Convolutional neural networks, autoencoders, and their related combinations, which are eligible under the umbrella of deep neural networks, have been employed to deal with the most disparate classification and detection tasks (e.g., unbalanced dataset, encrypted traffic, and anomaly detection) and their results are very promising [53–56].

Accordingly, in Chapter 4, we combine the abilities of Autoencoders in distilling meaningful and relevant features and the effectiveness of non-linear characteristics in capturing traffic behavior, arranged as Recurrence Plots, to introduce a detector capable of performing anomaly and attack detection/classification tasks in the presence of different conditions in terms of available training/testing datasets.

2.2 MALWARE DETECTION

Most of the existing malware are specifically designed to be self-modifying in order to evade pattern-matching detection mechanisms [57]. Such types of malware are also known as polymorphic and metamorphic malware, which can mutate the appearance of their code by adding several NOP and loop instructions, permuting user registers, and modifying static data structures. Consequently, static malware analysis-based approaches become ineffective and can be strongly affected by obfuscation tools. Therefore, to overcome these issues, several dynamic analysis-based methods have been proposed in the last years [58–61]. They analyze the dynamic behavior of an application by considering several aspects like function calls, parameters monitoring, and API calls tracing [62].

However, since most of them are characterized by a considerable computational complexity (i. e., they are often NP-Complete) [63, 64], several studies have adopted Markov chain-based models that represent the applications-related features vectors as transition probabilities between each API calls pair. Such approaches have shown their effectiveness against evasion techniques by proving that the insertion and removal of arbitrary calls do not significantly affect the transition probabilities [65, 66]. More precisely, in 2018, A. Martin et al. [9] presented a tool that classifies Android malware families by arranging the extracted dynamic features as a Markov chain. The following application, also known as Classifying Android malware families by modeling dynamic traces with Markov chains (CANDYMAN), has been validated on a collection of 4442 samples grouped into 24 different malware families by achieving a precision of 81.8%. In 2019, M. Ficco [8] proposed an IoT anomaly detector by considering the transaction probabilities associated with the API call sequences invoked by an application. In detail, the following model has obtained an F-measure of 89% on 22000 benign applications and 24000 malware collected from several famous datasets. D'Angelo et al. [67] presented a new associative rules-based malware classifier capable of combining the Markov chains and associative rules. More precisely, they have proven the effectiveness of the classification scheme on 8 malware families by comparing it with

the state-of-art algorithms and achieving an average F-score of 96.1%.

On the other hand, due to the recent explosion of malware attacks, also in scenarios characterizing new userland devices in the IoT segment market, and to the necessity of protecting the privacy and intellectual property at both the individual users' and organizations' level, the involved subjects are becoming increasingly reluctant to share their sensitive data and, consequently, being involved in any security scenario capable of defending them against new hostile activities. To overcome this issue by preserving privacy, one of the most popular options is associated with FL-based [68] solutions, where each decentralized entity trains an individual model using only its data and, in order to create a global and shared model, sends the obtained model parameters to a central server [69]. In this direction, many FL-based solutions have been proposed and adopted in several application domains, such as healthcare applications [70], Failure prognosis [71], and network traffic detection and classification [72, 73]. Also, they have been used for malware classification in IoT environments by considering several extracted features, FL algorithms optimization, learning models, and security schemes. Unfortunately, due to the high amount of related works published, performing a complete comparison among them is very difficult. A detailed comparison among the different solutions proposed in the literature is presented [73–75].

Ruei-Hau et al. [76] proposed a new Android detection schema to prevent possible Poisoning attacks on a federated model. They protected the federated learning process through a Secure Multi-Party Computation (SMPC) implementation provided by OpenMined/PySyft [77]. Moreover, they evaluated the discussed detection model in both centralized and decentralized scenarios by respectively obtaining an average accuracy of 94.05% and 93.45%. In the same direction, Galvez et al. [78] presented Less is More (LiM), an Android malware classification framework that leverages FL to detect and classify malicious applications by employing static features. More precisely, the following framework has achieved an average F-Score of 95% over 50 iteration rounds and 50000 Android applications distributed among 200 clients trained in a semi-supervised manner.

Shukla et al. [79] proposed a Robust and Active Protection with Intelligent Defense (RAPID) strategy against malicious activities based on CNNs and grey-scale figures representing the application binary files. The following malware classifier, able to distinguish 6 famous malware families, has proven its effectiveness by obtaining a 94% average accuracy and introducing, in addition, a server-side defense mechanism based on the euclidean distances among the related federated model features. Finally, in 2022, Rey et al. [6] and Popoola et al. [80] presented similar network flows-based approaches to detect the presence of Botnet-related cyberattacks in IoT domains with a 99% average accuracy. More precisely, the following methods have been trained by considering several learning scenarios (Supervised, Semi-Supervised, and Unsupervised), DNNs models (with different numbers of layers and hidden neurons), federated hyperparameters (number of clients and rounds), and network features respectively related to Bot-IoT and N-BaIoT datasets [81].

However, most of the reported FL-based approaches consider static features that, as previously remarked, are strongly affected by obfuscation techniques and polymorphic malware, while traffic flows-based ones, being based on features directly derived from the stored network packets, become ineffective against traffic anonymization techniques. Furthermore, since both of them are not able to analyze the application-related dynamic behavior, they cannot be used in any possible runtime-based detection or classification strategies [61].

Moreover, as previously highlighted in the introduction, FL-based models are often adversely affected by non-Independent and Identically Distributed (non-IID) data for what concerns training time, convergence, learning process, and classification results [82, 83]. For this reason, many learning strategies have been proposed in recent years. In 2020, Karimireddy et al. [20] presented the Stochastic Controlled Averaging algorithm (SCAFFOLD) that tries to estimate the update direction for both server and client models. More precisely, the proposed algorithm, validated in the presence of 100 clients and unbalanced dataset partitions, has outperformed the most famous state-of-the-art Federated algorithms by slightly improving the convergence process in terms of required computational time. Li et al. [84] proposed an extension

of the FedAvg algorithm based, at each iteration, on the selection of a subset of clients through their local loss function values. The presented FedProx algorithm, tested on four famous federated datasets, has achieved comparable IID and non-IID loss functions but at the expense of slower convergence.

Lu et al. [85] presented an improvement of the FedAvg algorithm based on Earth Mover's Distance (EMD) between central and local parameters. More precisely, a new metric, defined as node degree contribution, is derived to improve the local models-related aggregation at each iteration. Also, the experimental results carried out with 100 clients and a different number of iterations, have shown a similar convergence behavior and a slight improvement in accuracy compared with the FedAvg algorithm. Eventually, Pagliarola et al. [86] proposed a PartialNet Strategy that reduces communication costs by considering partially trained models that, at each iteration, are aggregated by the central server if and only if they satisfy a given threshold. The following strategy, validated on a real-world dataset regarding people affected by hypertension, has proven effective by reducing communication costs in both IID and non-IID data scenarios.

The discussed methods for malware detection tasks are summarized in Table 2.3.

Further, as previously stated, the proposed learning strategies are often characterized by additional hyperparameters whose tuning complexity could limit their applicability. Therefore, to overcome the discussed issues and preserve the intellectual property and privacy of the involved users and companies, we extend the associative rules-based detector, presented in [67], within a federated logic. More precisely, we recall the capability of our approach to be inherently robust against evasion/obfuscation techniques based on the API call flow perturbation. Also, we highlight the possibility of using it in non-IID data scenarios that, as previously discussed, often adversely affect the convergence and learning processes of the FL-based models.

Author(s)	Method(s)	Advantages
Martin et al. [9]	Markov chain, CANDYMAN	The privacy of the data can be preserved without losing model performance
Ficco [8]	Markov chain	The Markov chain approach detects unknown malware samples with F-measure up to 89%
D'Angelo et al. [67]	Association rule-based	The method is able to detect unknown malware samples with 99% accuracy and an F-measure of 96.10%
Ruei-Hau et al. [76]	FL, Support Vector Machine (SVM)	The privacy of app information and trained local models is guaranteed
Galvez et al. [78]	FL	The method is robust against both poisoning attacks by adversaries who control half of the clients, and inference attacks performed by an honest-but-curious cloud server
Shukla et al. [79]	FL, CNN	An average accuracy of 94% is obtained, while ensuring data security and privacy
Rey et al. [6]	FL ,DNN	The privacy of the data can be preserved without losing model performance
Popoola et al. [80]	FL, DNN	Detecting zero-day botnet attacks with high classification performance, having low communication overhead
Karimireddy et al. [20]	Stochastic Controlled Averaging algorithm	The method can take advantage of similarities in the client's data yielding even faster convergence
Li et al. [84]	Improved federated averaging	In highly heterogeneous settings, the method demonstrates significantly more stable and accurate convergence behavior
Lu et al. [85]	Federated averaging, EMD	The method can reduce the impact of non-IID data problems
Pagliarola et al. [86]	FL	The method reduces communication costs in both IID and non-IID data scenarios, The method achieves excellent results in terms of classification accuracy

Table 2.3: An overview of the discussed malware detectors.

3

PRELIMINARIES

The methods presented in this thesis utilize different machine learning and deep learning models. Moreover, different types of neural networks are employed to provide robust classifiers or detectors, which are suitable for most network security tasks. For the sake of clarity, in this chapter, the basic theoretical backgrounds of machine learning, deep learning, the employed neural networks, recurrence plots, and the Markov chain are described.

3.1 MACHINE LEARNING

(AI) has attracted lots of interest in recent years for many use cases, such as self-driving cars [87], chatbots [88], virtual assistants [89], and more [90]. The history of AI dates back to the 1950s when researchers aimed to automate intellectual tasks typically performed by humans. The dominant method for achieving human-level AI during this time was symbolic AI, which relied on a vast set of rules for manipulating knowledge. Even though symbolic AI successfully dealt with well-defined tasks, it proved limited in solving more complex tasks such as speech recognition and image classification. To overcome these challenges, the field of machine learning emerged as a new approach to AI.

The rise of Machine Learning (ML) has introduced a new approach to programming. In traditional programming and symbolic AI, rules and data are input by the user to produce results. In contrast, with ML, the user inputs data and desired results, and the learning model generates the rules. These rules are then applied to new data to produce outcomes. ML systems are trainable rather than explicitly programmable, meaning they require large amounts of data to identify meaningful features and generate rules for automation [91]. The process of building analytical models through explicit programming, shallow ML, and deep learning is illustrated in Figure 3.1.

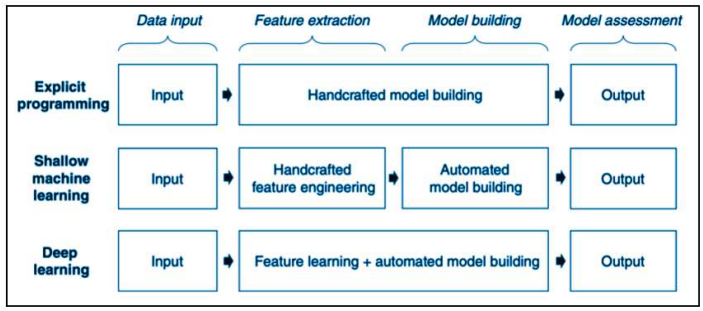

Figure 3.1: The process of analytical model building for explicit programming, shallow ML, and deep learning [92].

As a statistical-based analytical tool, ML has gained widespread use in various fields such as healthcare, finance, marketing, and cybersecurity, among others. Its ability to make informed decisions based on data analysis relieves the burden of processing vast amounts of information, making it ideal for analyzing complex situations. Additionally, its ability to respond quickly to abnormal behavior is an advantage in early detection, outpacing human response times [93].

ML is commonly classified into three main learning paradigms: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning (as shown in Figure 3.2). These paradigms vary in the types of tasks they can solve and how data is presented to the computer. The task and data often dictate which paradigm should be used, with supervised learning being the most common. However, in some cases, multiple paradigms can be combined to achieve better results [94]. This section provides an overview of these paradigms and their use cases.

Supervised Learning (SL). Supervised Machine Learning uses labeled training data to create a target model (function). A labeled data consists of an input feature vector (x) and an output label (y). The output label serves as the supervisory signal. The learning algorithm learns the relationship between the feature vector and label in the training data and applies that knowledge to new data to accurately predict class labels.

Therefore, in SL, the objective is to learn a target function (model) from the dataset to the set of labels using a training set,

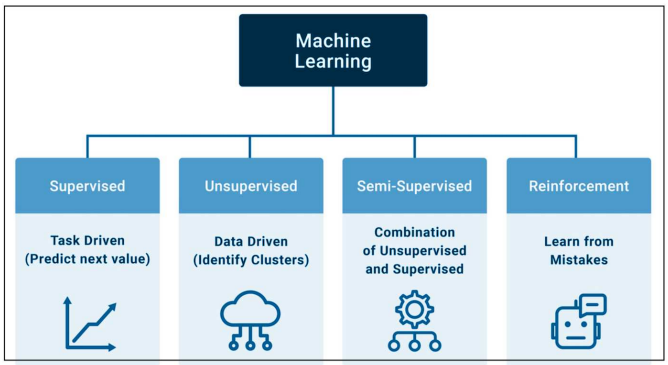

Figure 3.2: ML main learning paradigms.

which is a subset of the dataset. Past experience is used as a reference for decision-making, and a high-quality training set is crucial for building an effective model. However, a successful outcome is not solely dependent on the dataset; the training method also plays a significant role. Before making predictions, the model's performance, such as accuracy, is usually evaluated to determine its reliability. The model's effectiveness is determined by its predictive performance on test data, another subset of the dataset.

SL can further be classified into two techniques: classification and regression. Classification involves categorizing input data into discrete groups based on the probabilities of each group belonging to a certain class. The class with the highest likelihood of belonging to the sample wins [95]. On the other hand, regression predicts continuous responses from input variables. For example, it can predict the temperature or demand for power [96]. The percentage of correct predictions measures the performance of a classification model. In contrast, a regression model's performance is evaluated by calculating the root-mean-square error, which measures the deviation between the predicted and actual values.

Some examples of SL algorithms are as follows:

- Support Vector Machines (SVM) for classification;
- decision trees for classification;
- Naive Bayes for classification;

- K-nearest neighbors (K-NN) for classification;
- random forest for classification and regression;
- linear, logistic, polynomial, and least square regression for regression;
- ensemble methods for classification and regression.

Unsupervised Learning (UL). Unsupervised machine learning is a process of learning without the use of labeled training data. The input data (x) is the only information available, with no corresponding output data (y) for training. In this scenario, unsupervised learning algorithms attempt to identify the structure and distribution of the data to uncover and present any significant patterns or classes present. This type of learning is referred to as unsupervised because there is no supervisor providing the correct answers.

In UL, unlike SL, there is no labeled training data and a training process. This means the system operates independently, and its performance is difficult to evaluate. While some researchers use pre-existing labeled data to assess the results of the UL model, this is not practical in real-world applications, and experts may need to manually analyze the results to perform an external evaluation [94].

The main objectives of UL are to discover patterns and relationships in data, estimate the probability distribution that generated the data, find groups in the data through clustering, identify associations in the data, detect outliers, and reduce the complexity of the data through dimensionality reduction. Clustering aims to identify natural groupings in the data, while association rule learning seeks to uncover rules that describe a significant portion of the data. Through dimensionality reduction, unsupervised learning identifies related features in the dataset, allowing for the removal of redundant information to minimize noise.

Some examples of UL algorithms are as follows:

- hierarchical clustering for clustering;
- mixture models for clustering;
- K-means clustering for clustering;

- factor analysis for clustering;
- Principal Component Analysis (PCA);
- apriori algorithm for association problems;
- independent component analysis;
- singular value decomposition.

Semi-Supervised Learning (SSL). Semi-supervised machine learning is used when the training data consists of a small number of labeled points and a large number of unlabeled points. This can happen due to the limited availability of labeled data or its high cost of production. These problems bridge the gap between supervised and unsupervised learning. Collecting and storing unlabeled data is usually inexpensive and straightforward, but obtaining labeled data is time-consuming and costly. Many practical machine learning scenarios fall under the umbrella of SSL.

SSL uses the supervised learning process with limited labeled data to develop a classifier that can predict the class of unlabeled samples. The classifier then assigns a confidence score to each pseudo-labeled sample, allowing the administrator to determine its accuracy. The confident pseudo-labeled samples are added to the training set, and the classifier is updated until all data is labeled. However, because the pseudo-labeling is done randomly, certain assumptions such as smoothness and clustering must be made before training on the unlabeled data [97, 98].

Therefore, SSL can also be seen as learning with constraints, where constraints are imposed on the behavior of the learned classifier on unlabeled instances. SSL can take on either an inductive or transductive approach. In the inductive approach, the unknown function is inferred from the available data, while in the transductive approach, the values of the unknown function for specific points of interest are inferred from the available data.

Some types of SSL techniques are as follows:

- self-labelled techniques;
- semi-supervised support vector machines;
- generative models.

- graph-based models;

Reinforcement Learning (RL). RL is a powerful machine learning technique that has gained significant attention recently due to its ability to solve complex problems in dynamic environments. Unlike supervised learning, which requires labeled data, and unsupervised learning, which requires only unlabeled data, RL requires an interactive environment where an agent can receive feedback through rewards. Indeed, in RL, an agent learns from its interactions with the environment by continuously adjusting its behavior to maximize the expected cumulative reward. The learning process is based on trial and error, where the agent takes action, receives rewards, and updates its policy. The agent's goal is to find the optimal policy that maximizes the reward over time [99].

RL has been applied in various fields such as robotics, gaming, and control systems. RL has been used in robotics to train robots to perform tasks such as grasping objects, navigation, and manipulation. RL has been used in gaming to develop AI agents that can compete with human players in complex games such as chess, Go, and video games. RL has been used in control systems to optimize control policies for systems such as energy management and traffic control systems.

RL algorithms can be classified into value-based, policy-based, and actor-critic algorithms. Value-based algorithms aim to estimate the expected reward for each state or state-action pair. Policy-based algorithms focus on directly learning the optimal policy, while actor-critic algorithms combine value-based and policy-based methods.

In conclusion, RL is a promising machine learning technique that offers a new way of solving problems in dynamic environments. It has the potential to be applied in a wide range of fields and has already been used to develop advanced AI systems in various domains.

Some categories of RL algorithms and techniques are as follows:

- Monte Carlo;
- Q-learning (State-Action-Reward-State);

- State-Action-Reward-State-Action (SARSA);
- Q-Learning-Lambda (State-Action-Reward-State with eligibility traces);
- SARSA-Lambda (State-Action-Reward-State-Action with eligibility traces);
- Trust Region Policy Optimization (TRPO);
- Deep Deterministic Policy Gradient (DDPG);
- Deep Q network (DQN);
- Proximal Policy Optimization (PPO).

3.2 DEEP LEARNING AND NEURAL NETWORKS

Traditional ML methods faced challenges in processing natural data in its raw form. Building a pattern recognition or ML system was a complex and time-consuming process that required extensive domain knowledge and engineering expertise. The raw data, such as the pixel values in an image, needed to be transformed into a suitable internal representation or feature vector through a feature extractor before the learning subsystem, usually a classifier, could recognize or classify patterns in the input.

Representation learning is a branch of machine learning that enables machines to learn the representations necessary for pattern recognition and classification from raw data. This is achieved through the use of novel learning methods that consist of multiple levels of representation created by composing simple but non-linear modules. Each module transforms the representation from one level to a higher and more abstract level. The composition of enough such transformations enables the learning of very complex functions. In classification tasks, higher representation levels highlight important input features for discrimination and minimize irrelevant variations. The key characteristic of this approach is that the feature extraction layers are not manually designed by humans but learned from data using a general-purpose learning algorithm [100].

Deep Learning (DL), a subfield of ML and AI, is considered a cornerstone technology of the current Fourth Industrial Revolution (4IR or Industry 4.0) (Figure 3.3). DL methods utilize representation learning to model data through complex architectures that incorporate various non-linear transformations. The building blocks of DL are neural networks, which are combined to form deep neural networks. Because of its ability to learn from data, DL, which evolved from artificial neural networks, has gained significant attention in the field of computing and has been widely applied in various fields, including visual recognition [101, 102], speech recognition [103], natural language processing [104], language translation [105, 106], healthcare [107], cybersecurity [108–111], and many more.

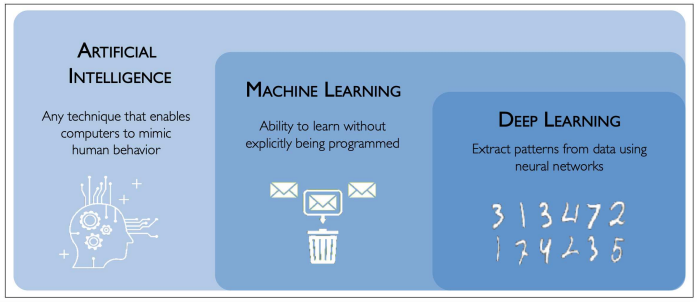

Figure 3.3: The hierarchical relationship between AI concepts and classes.

Neural networks, the building blocks of deep learning models, have garnered increasing attention in research since the late 1980s. They provide a potent set of solutions for specific problems that are complementary to those of more conventional techniques. An Artificial Neural Network (ANN), often known as Neural Network (NN), is a mathematical model designed to emulate the structure and function of biological neural networks. From the mathematical point of view, ANN is an application f, which is non-linear concerning its parameters θ that associates to an entry x an output $y = f(x, \theta)$. As usual in statistical learning, the parameters θ are estimated from learning samples. Every ANN is constructed using its basic blocks, artificial neurons, which are simple mathematical (or computational) models. These models function using three basic operations: multiplication, summation,

and activation. These three operations define the core rules of the model. The first step in processing inputs in an artificial neuron is to be weighted, which involves multiplying each input value by a corresponding weight. The weighted inputs and a bias term are then summed together. Finally, the output is obtained by passing the sum value through an activation function (Figure 3.4).

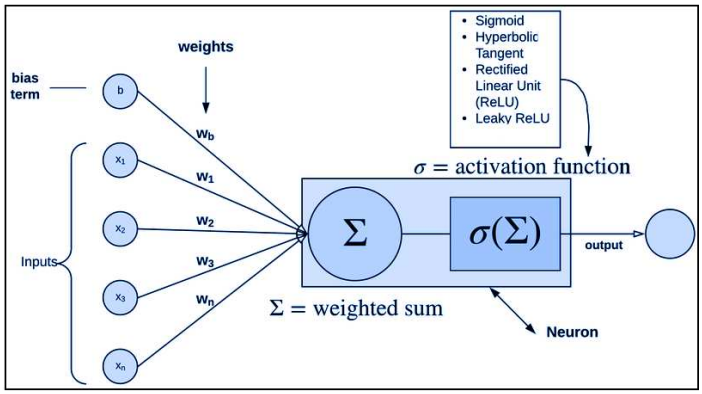

Figure 3.4: General model of an artificial neuron.

Several types of architectures exist for ANNs, such as deep neural networks (multilayer perceptrons), convolutional neural networks, recurrent neural networks, autoencoders, and so on. In the following, some of the most important types of these architectures that are used in this thesis are explained in detail.

3.2.1 *Deep Neural Networks*

Deep Neural Network (DNN) is a type of neural network with multiple hidden layers and neurons. These networks have the ability to learn essential features, leading to improved classification and prediction performance. Because of their ability to build strong prediction models and adapt to non-linear situations, DNNs are being widely used in various fields [112]. They are also known as fully-connected feed-forward neural networks and can be depicted as a Directed Acyclic Graph (DAG) with data flowing from the input layer to the output layer in a single direction (Figure 3.5) [100]. The architecture of DNNs can be

changed by adjusting the hyper-parameters during the training phase [92].

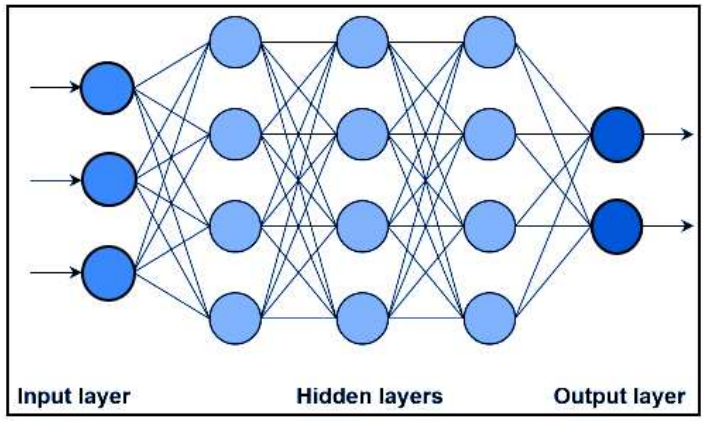

Figure 3.5: Typical structure of the DNN model.

We assume an input data vector x and a standard DNN network. Given these settings, the DNN carries out the following operation:

$$y = \sigma(W.x + b). \tag{3.1}$$

In the equation, y represents the output from a layer, W stands for the weights being learned, and b refers to the bias neurons. $\sigma(.)$ is an activation function that enhances the training process of the model by adding non-linearity to it. The most frequently used non-linear activation functions are depicted in Figure 3.6.

The ReLU and Leaky ReLU activation functions have been introduced as a solution to the issue of gradient vanishing in other activation functions. Gradient vanishing occurs when the gradients of the loss function become extremely small, making it difficult for them to pass through the layers. This problem has been addressed in [113].

3.2.2 *Convolutional Neural Networks*

Convolutional Neural Network (CNN) is considered to be one of the most efficient types of deep neural networks and has proven to be particularly effective in fields such as image processing,

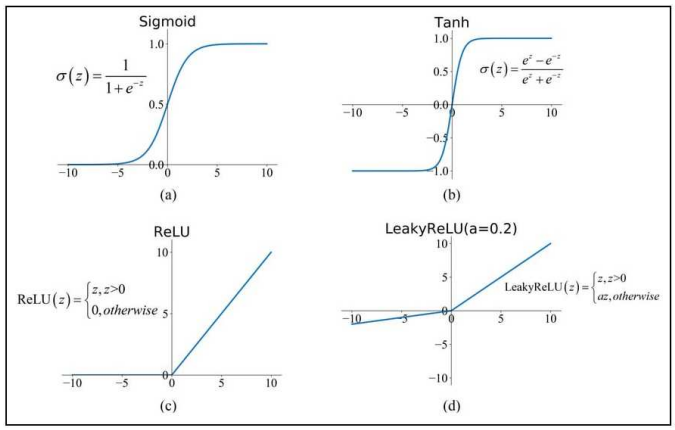

Figure 3.6: Commonly used activation functions.

object recognition, natural language processing, and autonomous driving. CNNs can be applied to various types of data, including time-series data which can be represented as a 1-dimensional grid of regularly sampled data points; image data which can be represented as a 2-dimensional grid of pixels; and video data which can be viewed as 3-dimensional tensors. This versatility and practical success is why CNNs are widely used. The term convolutional in CNN refers to the use of a mathematical operation known as convolution.

$$s(t) = (x * w)(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da. \tag{3.2}$$

In the context of convolutional networks, the first input to the convolution operation (in Equation 3.2, the function x) is typically referred to as the input, while the second input (in Equation 3.2, the function w) is known as the filter or kernel. The result of the convolution operation is often referred to as the feature map. In computer science, the convolution operation is typically expressed in its discrete form as follows:

$$s(t) = (x * w)(t) = \sum_{-\infty}^{\infty} x(a)w(t-a). \tag{3.3}$$

In CNNs, the input is typically a multi-dimensional array of data, often referred to as a tensor, while the kernel or filter is

another tensor consisting of parameters that are learned by the algorithm. As each element of the input and kernel must be stored individually, it is usually assumed that these functions are zero everywhere except for the finite set of points with stored values. This means that the infinite summation in the convolution operation can be computed as a summation over a finite number of elements in the arrays.

Convolution operations are often performed across multiple axes simultaneously. For instance, if the input is a 2-dimensional image, a 2-dimensional kernel K is often used:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \quad (3.4)$$

Convolution is commutative, meaning that it can be expressed equivalently as:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \quad (3.5)$$

The popularity of using CNNs for deep learning has grown significantly due to two main reasons:

- the first reason is that CNNs eliminate the need for manual feature extraction, as the network can learn these features directly.
- the second reason is that CNNs can be retrained for new recognition tasks, enabling leveraging existing networks.

Like other neural networks, CNNs have input and output layers. Their strength is the multiple hidden layers, which use specific mathematical operations to learn relevant features. Lower layers learn basic features (Low-Level features), while higher layers use these basic features to learn more complex and abstracted features (High-Level features). Typically, each layer performs the following three operations in sequence:

- **Convolution.** The main operation in a CNN is convolution. It applies multiple convolutional filters to the input data, each filter designed to detect specific features. The sliding involved in the convolution is performed by different artificial neurons that share the same weights.

- **Rectified Linear activation fUnction (ReLU).** The ReLU function, as stated, outputs zero for negative input values and the input itself for positive input values. It is utilized as the activation function for the output of each convolutional neuron. This function has been a breakthrough in deep learning due to its ability to tackle the vanishing gradient problem, a well-known issue, and achieve improved learning performance. Unlike the commonly used sigmoid and hyperbolic tangent activation functions, ReLU has proven to be more effective.

- **Pooling.** Pooling is utilized as a down-sampling function for two primary reasons: to decrease the size of the output layers and thus the number of connections with subsequent layers and to make the features insensitive to location. This leads to faster learning and allows CNNs to recognize features regardless of their position (achieving local translation invariance). The two most common pooling methods are max-pooling and average pooling, with the former accentuating the most active feature and the latter highlighting the average value of the features.

The final step in implementing a CNN is the classification process, which uses a fully connected feed-forward neural network with an output vector equal in size to the number of classes to be predicted. The softmax function is typically used to classify based on probability values (Figure 3.7).

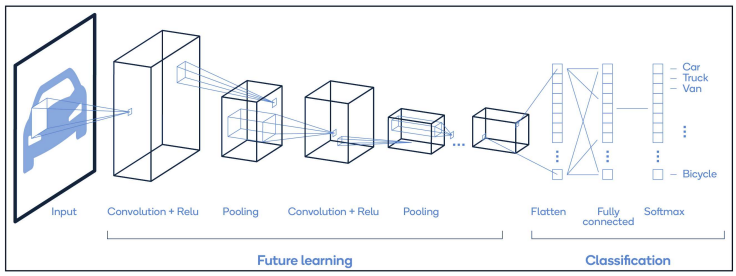

Figure 3.7: General architecture of a CNN.

3.2.3 *Recurrent Neural Networks*

Feed-forward neural networks return almost identical outputs for the same inputs, which can be desirable in some cases but limiting in others where the network needs to respond differently based on past inputs. For this purpose, the network needs to preserve the memories of the past. This may be done by providing a state to the network and using it as part of the subsequent operations.

Recurrent Neural Network (RNN) is a popular neural network model that is widely used to analyze sequential data. The feedback loops in RNNs have a significant impact on the network's behavior even for the same inputs, making it a suitable choice for analyzing sequences [114]. RNNs differ from CNNs in their design and purpose. CNNs are optimized for processing grid-like data, such as images, while RNNs are specialized in working with sequences of values like $x_1, x_2, ..., x_t$. Additionally, RNNs can handle sequences of varying lengths. Recurrent networks and some other machine learning and statistical methods utilize the innovative approach of sharing parameters across different layers of the model, allowing the model to be used for data instances with varying forms. Parameter sharing is especially crucial when a particular item of data may appear at multiple positions within the sequence. This optimization technique can result in significant memory savings in machine learning models [115]. The main advantage of using RNNs over traditional neural networks is their ability to handle sequential data, where each sample is dependent on previous ones.

As mentioned, RNNs are specifically designed to model sequences where there is a strong correlation between consecutive samples. At each time step, RNN combines the current input and the state, which contains information from previous steps, to produce output. This information is passed through recurrent connections between units, as illustrated in Figure 3.8. Given a sequence of inputs $\mathbf{x} = (x_1, x_2, ..., x_t)$, an RNN performs the following computations:

$$s_t = \sigma_s(W_x x_t + W_s s_{t-1} + b_s), \tag{3.6}$$

$$h_t = \sigma_h(W_h s_t + b_h), \tag{3.7}$$

where s_t represents the state of the RNN at time step t, serving as a memory unit. The value of s_t is computed as a function of the input at time t, (x_t), and the previous state of the RNN, s_{t-1}. Moreover, the weights W_x and W_h are learned during training, while b_s and b_h represent the biases. The training of an RNN is

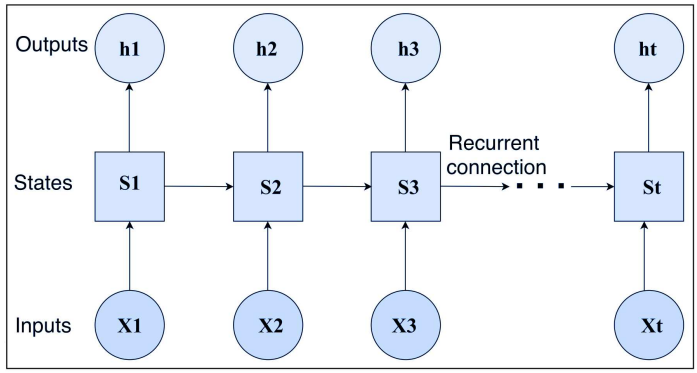

Figure 3.8: General structure of an RNN.

done using a variation of the Backpropagation algorithm, the Backpropagation Through Time (BPTT) algorithm [116]. BPTT involves unrolling the RNN into a series of feed-forward neural networks, leading to a deep network with multiple hidden layers, which is then trained through the Backpropagation algorithm. However, this process is susceptible to the vanishing gradient problem. To mitigate this issue, the long short-term memory network was introduced [117].

The core idea of the Long Short-Term Memory (LSTM) model is to use self-loops to store the gradient of recent input events for long durations [117]. This allows them to effectively learn and model long-term dependencies in sequential data, making them useful for tasks such as handwriting recognition [118], speech recognition [119], handwriting generation [120], machine translation [121], image captioning [122], and parsing [123]. LSTM was developed to tackle two significant issues present in previous techniques, namely, vanishing and exploding gradients. Conventional gradient-based learning methods such as BPTT

and Real-Time Recurrent Learning (RTRL) can cause error signals to decrease or increase during back-propagation through the model. The LSTM network was designed to address the problems of back-propagation of error signals. This was achieved by incorporating a system of gates into the network. A graphical illustration of the structure of an LSTM network is shown in Figure 3.9. The LSTM structure has an Internal Cell State (c_t)

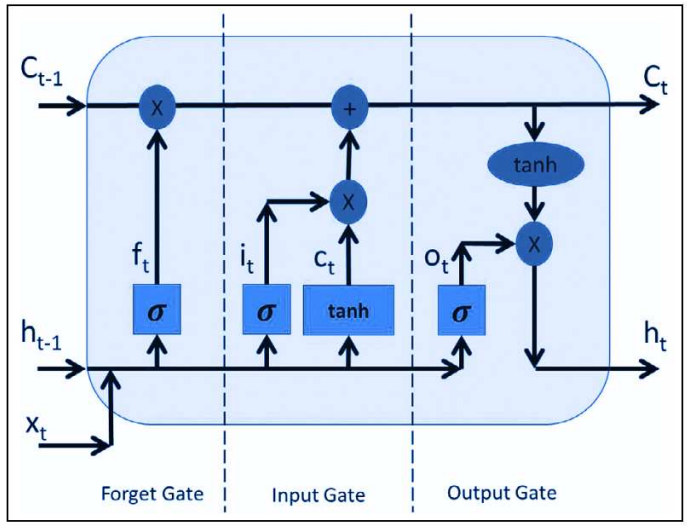

Figure 3.9: Schematic representation of an LSTM cell.

that enables it to keep or discard previous information. The state is updated by four internal activation layers, known as gates, which are constructed using a sigmoid neural network layer and a point-wise operation. The purpose of each gate is defined as follows:

- Forget Gate (f_t): decides the amount of past information that needs to be forgotten.
- Input Gate: determines the amount of information from the current input to be stored in the cell state. It combines the output from a sigmoid layer (i_t) and a *tanh* layer (c_t) to make the decision. The sigmoid layer decides which information to update, and the *tanh* layer decides the magnitude of the chosen information to be added to the current state.

- Output Gate (o_t): determines which and how much data to provide as output (h_t) starting from the actual cell state (c_t).

3.2.4 *Autoencoders*

Autoencoders (AEs) are artificial neural networks that operate without supervision to generate new data through a sequence of two processes, encoding and decoding. As depicted in Figure 3.10, the first process is implemented by an encoder which is devoted to compressing the input into a space of latent variables. In contrast, the second process is implemented by a decoder which is involved in reconstructing the input based on the information included in latent variables. Both the encoder and decoder are typically implemented as neural networks, with the encoder and decoder networks being usually symmetric to each other in terms of their architecture and weight matrices.

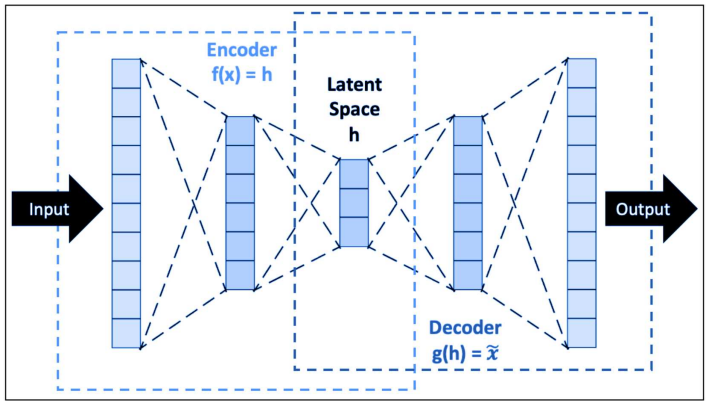

Figure 3.10: Schematic representation of an Autoencoder.

The principal purpose of AE is to learn the key features of its input data so that it can copy the input as the desired output. By copying only the most important and representative aspects of the input, the AE can identify the most useful properties of the data and use them to reconstruct the original data. From a mathematical point of view, assume there is a training set of $\{x^1, x^2, x^3, ..., x^n\}$ where for each data sample, we have $x^i \in \mathbb{R}^n$. The objective of the AE is to reconstruct the network input by reducing the reconstruction error, i.e., $\tilde{x}^i = x^i$ for $i \in \{1, 2, 3, ..., n\}$.

To put it simply, the AE attempts to learn a compressed representation of the input data. Given this objective, the AE endeavors to minimize the following loss function:

$$\Gamma(W, b) = ||x - F_{W,b}(x)||^2, \tag{3.8}$$

in which W and b are the vectors of the network weights and biases, respectively, and $F_{W,b}(x)$ is the identity function that the AE tries to learn.

Moreover, AEs are used to create a latent space with a dimensional smaller than the original data dimension in a manner similar to techniques for data compression through dimensionality reduction like Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), Discriminant Function Analysis (DFA), and t-distributed Stochastic Neighbor Embedding (t-SNE) [124]. The crucial difference between AEs and these dimensionality reduction techniques is that AEs are capable of representing input data using non-linear combinations of features. This enables them to represent more intricate input data with a low-dimensional latent space.

The specific type of neural network employed in the encoder-decoder pair determines the overall functionality of the Autoencoder (AE). Basically, there are seven types of AEs, Sparse AE (SAE) [125], Contractive AE [126], Denoising AE [127], Undercomplete AE [128], Deep AE [129], Variational AE [130], and Convolutional AE [131].

3.2.5 *Stacked Neural Networks*

Constructing robust and reliable neural network models for a classification task can be challenging and require a significant amount of time. However, their performance can be improved by merging and combining multiple related models into a single Stacked Neural Network (SNN). The theory behind SNNs is based on the idea that combining multiple feature types, each learned by a different network, reduces uncertainty and improves the balance between the speed of the training process and the accuracy of classification [132, 133]. The training process of an SSN starts by training each individual model independently and then fine-tuning the entire network through a supervised method, such as

backpropagation (Figure 3.11). This method helps to overcome the vanishing gradient problem.

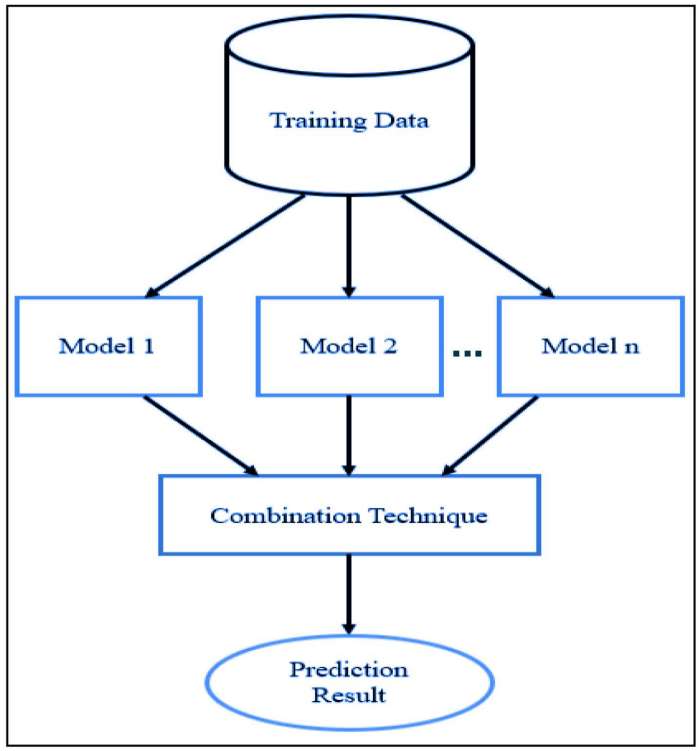

Figure 3.11: General architecture of an SNN.

Additionally, SNNs have become increasingly popular in the context of Autoencoders, which are referred to as Stacked Autoencoders. Stacked Autoencoders have been demonstrated to be highly effective in handling unbalanced and noisy datasets [127, 134]. A Stacked Autoencoder is implemented by connecting the encoders of multiple AEs in sequence, followed by a fully connected network.

3.3 RECURRENCE PLOTS

Many methods for classification and detection tasks begin by investigating particular non-linear features, such as recurring events and concealed non-stationary patterns in the time series related to the traffic classes that need to be explicitly distin-

guished. A preliminary study, also known as the supervised learning phase, is necessary to identify the most distinguishing characteristics of each type of traffic flow using pre-classified reference flows. This is a time-consuming and challenging process that requires significant computing resources and human expertise. However, it only needs to be completed once during the initial 'knowledge construction' phase of the model. Once the qualitative discrimination methods have been established, all subsequent tasks include a 'quantitative' recurrence assessment that can be realistically performed online for specific purposes and hence resource-limited network devices. The preliminary study mentioned above can be carried out through the use of Recurrence plots (RPs) analysis. This method provides valuable information about the non-stationary patterns of variation in time-series data. The core concept is to reconstruct the dynamics of an unknown system in the phase space by using time-delay embedding and then calculating the distances between all pairs of embedded vectors, resulting in a symmetric two-dimensional matrix. The RP visualizes and represents this distance matrix.

3.3.1 *Non-Stationarity Characteristic*

Stationarity is extremely important in the context of traditional linear and non-linear time series analysis, mainly when working with time series of traffic variables which are often non-stationary [135]. The term "stationarity" refers to an assumed regularity pattern in a data series [136], and a time series is considered non-stationary when the joint probability distribution of $x_i, x_{i+1}, ..., x_{i+(q-1)}$ depends on the time index i for some value of q [137]. Traffic volume analysis in traffic engineering is closely related to the concept of non-stationarity. Detecting non-stationarity is crucial because it identifies changes in the temporal statistical behavior of the underlying process. It is important to identify these points and changes in many dynamic phenomena [135]. Sudden changes in the statistical characteristics of traffic variables, such as volume or packet size, can provide insight into the different dynamics associated with the specific behavior of the involved end applications.

3.3.2 *Reconstructing the Phase Space: Delay-Coordinate Embedding*

A comprehensive analysis of a dynamic system is usually possible when the equations of motion and all the degrees of freedom (n) are known. The evolving state of such a dynamic system can be represented by sequences of "state vectors" or vectors of state variables in the phase space [138]. Unfortunately, only a limited number of quantities can usually be observed in a system. However, it is possible to reconstruct the whole dynamics of the system from a relatively small set of observable variables. It is not common to have complete knowledge of all state variables in the analysis of a complex non-linear system like an end-to-end traffic flow, as the system's evolution is shaped by the interaction of a very complex set of such variables. The analyst has access to only one time series, which is obtained through sampling from a single observation point.

The delay-coordinate embedding method employs past values to construct a proper representation of the internal dynamics. Takens theorem states that by using the time delay method, it is possible to recreate a topologically equivalent representation of the behavior of the original multi-dimensional system (In other words, the theorem implies that the phase space trajectory can be reconstructed) using the time series of a single observable variable: starting from the scalar time series $\{x_t\}_{t=1}^{T}$ we construct a sequence of (embedded) vectors $\mathbf{y}_{(i)} = (x_i, x_{i+\tau}, x_{i+2\tau}, ..., x_{i+(m-1)\tau})$ [139–141]. The set of all embedded vectors $\mathbf{y}_{(i)}$, $i = 1, ..., T(m-1)\tau$, establishs a trajectory in $\Re^m$ where m is the embedding dimension, and τ is the time delay. Each point of the phase space of the system at time i is reconstructed using the delayed vector $\mathbf{y}_{(i)}$, which is placed in an m-dimensional space referred to as the reconstructed phase space. According to the Takens theorem, the sequence of embedded vectors that recreate the original dynamics must have the correct values of m and τ. In particular, the value of m must be greater than $2d + 1$, where d is the dimension of the original, unknown system. The Correlation Dimension D_2 of Grassberger–Procaccia can be used to estimate the original dimension d of the system [36].

Therefore, choosing an appropriate value for the time delay τ and the embedding dimension m is a crucial step in delay-coordinate embedding. Several methods have been developed to make an informed guess, such as the Average Mutual Information (AMI) function for determining τ [142], and the False Nearest Neighbours (FNN) method for determining m [143]. These methods have been widely used in the literature and have been shown to provide good results.

The AMI minimum is considered a good estimate for τ as it is based on the idea that variables that are uncorrelated produce values that are also uncorrelated. Suppose that the time series domain is divided into equiprobable bins. Let p_i be the probability of having a time series value in the *ith* bin, let $p_{i,j}(\tau)$ be the joint probability to have a time series value in the *ith* bin and a time series value in the *jth* bin after a time τ, that is the probability of transition from the *ith* to the *jth* bin in τ time. The average mutual information function is:

$$S(\tau) = \sum_{i,j} p_{ij}(\tau) ln(\frac{p_{ij}(\tau)}{p_i p_j}). \tag{3.9}$$

The proposal to use the first zero-crossing of the autocorrelation function is supported by a similar argument [36]. Some other authors propose using the first maximum of the AMI instead, as it corresponds to the system's inherent periods. One effective method to determine suitable values for m is through the use of false nearest neighbors. This method of determining the embedding dimension through false nearest neighbors examines the number of false nearest neighbors with respect to m. False nearest neighbors are data points that were closest in lower embedding dimensions but become separated as the dimension is increased. For example, two points on a circle may appear close to each other even though they are not if the circle is viewed sideways (as a projection), resulting in the appearance of a line segment. Increasing the dimension m of the reconstructed space by one allows distinguishing between the orbital points, i.e., those that are true neighbors and those that are not. Let $\mathbf{y}$ be a point of the reconstructed space. Let $\mathbf{y}^{(r)}$ be the *rth* nearest neighbour of $\mathbf{y}$

and compute the (squared) Euclidean distance D^2 between them (as usual, y_k denotes the kth component of $\mathbf{y}$)

$$D_m^2(\mathbf{y}, \mathbf{y}^{(r)}) = \sum_{k=1}^{m-1} |y_k - y_k^{(r)}|^2. \tag{3.10}$$

Next, raise m to $m+1$ and calculate the new distance, i.e., $D_{m+1}^2(\mathbf{y}, \mathbf{y}^{(r)})$. The point $\mathbf{y}^{(r)}$ is said to be a false nearest neighbor if:

$$\frac{D_{m+1}^2(\mathbf{y}, \mathbf{y}^{(r)}) - D_m^2(\mathbf{y}, \mathbf{y}^{(r)})}{D_m^2(\mathbf{y}, \mathbf{y}^{(r)})} > D_{TS}. \tag{3.11}$$

where D_{TS} is a predetermined threshold. Keep in mind that the number of false nearest neighbors is dependent on D_{TS}. In practice, the percentage of FNN is calculated for each m of a set of values. The embedding dimension is said to be found for the first m such that the percentage of FNN falls to zero. Because this percentage never reaches zero with real-world and noisy data, the embedding dimension with the lowest FNN percentage is usually chosen.

3.3.3 *Building the Recurrence Plots*

The next step in RP analysis is to compute the mutual distances between embedded vectors in order to construct the recurrence plot. As a result, a standard must be chosen. The L_1-norm (minimum norm), the L_2-norm (Euclidean norm), and the L_∞-norm (maximum norm) are three commonly used norms, according to [36].

An RP is essentially a two-dimensional graphic representation of the distances matrix $\mathbf{D} = \{d_{i,j}\}$, where the pixel at coordinates (i, j) is shaded based on the distance between the ith and jth vectors. More accurately, if the distance $d_{i,j}$ is less than a predefined cutoff value ϵ (the two points are sufficiently near to each other), a dot is plotted in (i, j). Because each coordinate i represents a point in time, RP provides information about the temporal correlation of phase space points.

As a matter of fact, in RP, each horizontal coordinate i relates to the system's state at i, and each vertical coordinate j relates to the

system's state at j. Therefore, a recurrent point in (i, j) indicates that the interaction between the observed quantities is recurring, as it is almost the same in both instants i and j. Thus, if the point (i, j) is characterized as recurrent, the state j belongs to the neighborhood of size ϵ centered on i. This signifies that the system's state at i is comparable to the system's state at j; in other words, the system stays in neighboring orbits. Formally, the following equation describes the recurrence of a state x from the time i in a different time j [144]:

$$r_{i,j} = \theta(\epsilon - ||\mathbf{x}_i - \mathbf{x}_j||), \qquad i, j = 1, 2, ..., N, \tag{3.12}$$

where $r_{i,j}$ is an element of the recurrence matrix $\mathbf{R}$, N is the number of states x_i in the time window of the study, ϵ is the threshold for the distances, $||.||$ is a norm, and $\theta(x)$ is the Heaviside function, defined as $\theta(x) = 0$, for $x < 0$, $\theta(x) = 1$, for $x \geq 0$. In other words, $r_{i,j}$ gives a value of 1 if (i, j) is recurrent and a value of 0 otherwise. Note that the RP is symmetric (assuming a constant value of ϵ), i.e., $D_{i,j} = D_{j,i}$. Moreover, as $r_{i,i} = 1$ (for $i = 1, 2, 3, ..., N$), an RP will always include a diagonal line angled at 45° called the Line of Identity (LOI).

Every recurrence point represents an isolated recurrence of the phase relationship between the time series. Line segments that are parallel to the main diagonal originate from points near each other successively forward in time $(i, j), (i + 1, j + 1), ..., (i + l, j + l)$ such that $\mathbf{x}_j, \mathbf{x}_{j+1}, ..., \mathbf{x}_{j+l}$ is (respectively) close to $\mathbf{x}_i, \mathbf{x}_{i+1}, ..., \mathbf{x}_{i+l}$. Therefore, a diagonal line denotes a stable recurrence of the phase relationship during the time period corresponding to the diagonal's length (l). The recurrence period is the time interval that separates different diagonals. Given that they are not sequentially forward in time, recurrent points arranged in rows and columns do not provide any information regarding the timing of the periodicity. Nevertheless, horizontal and vertical lines are connected with stationary states. The orbit in the phase space will circle around a few points in the future and create a representation of the system's attractor if the time series is deterministic. The RP will then show short upward line segments parallel to the principal diagonal. Those segments correspond to sequences. In contrast, the RP of a completely random sequence will not exhibit any structure at all.

The goal of RP analysis is to find a number of patterns (in the RP) that point to statistical characteristics of the time series, including non-stationarity, data drifts, etc. A patterned RP is produced by any non-stationary process. The RP will show a fading toward the corners if the procedure has a pattern. The RP includes interruptions (dark-colored bands) in the case of abrupt changes. In the RP, diagonal lines and a checkerboard structure result from the cyclicity of an oscillating process.

On a more detailed and intricate level, there are some fundamental patterns, frequently referred to as small-scale structures, that are strictly related to deterministic structure and non-linearity. These patterns include single dots, diagonal lines, and vertical and horizontal lines (the combination of vertical and horizontal lines obviously forms rectangular clusters). The quantitative analysis of the RPs is based on these small-scale structures.

For instance, horizontal or vertical lines on an RP (($r_{i,j+k} = 1$ for $k = 1, \ldots, v$, where v is the length of the vertical line) indicate that the system state either does not change or changes extremely slowly over time. The system seems to have been stuck in a state for some time. This is a common characteristic of laminar states known as intermittency.

Diagonal lines ($r_{i+k,j+k} = 1$ for $k = 1, \ldots, l$, where l is the diagonal line's length) represent trajectories that pass through the same area of the phase space at various time periods. As a result, when the series exhibits any determinism or periodicity, parallel and perpendicular lines to the main diagonal appear. More particularly, when states evolve deterministically at various intervals, diagonal lines parallel to the LOI appear. Phase discordances are represented as diagonal structures parallel to the LOI (this is often a hint for an inappropriate embedding). The lengths of diagonal lines are directly related to the ratio of determinism or predictability inherent to the system. Assume that the states are neighbors at periods I and j, or that $r_{i,j} = 1$. Similar circumstances result in a similar future, which means the probability for $r_{i+1,j+1} = 1$ if the system behaves predictably. This results in infinitely long diagonal lines for systems that are completely predictable (like in the RP of the sine function). In contrast, if the system is stochastic, the probability of $r_{i+1,j+1} = 1$ is very

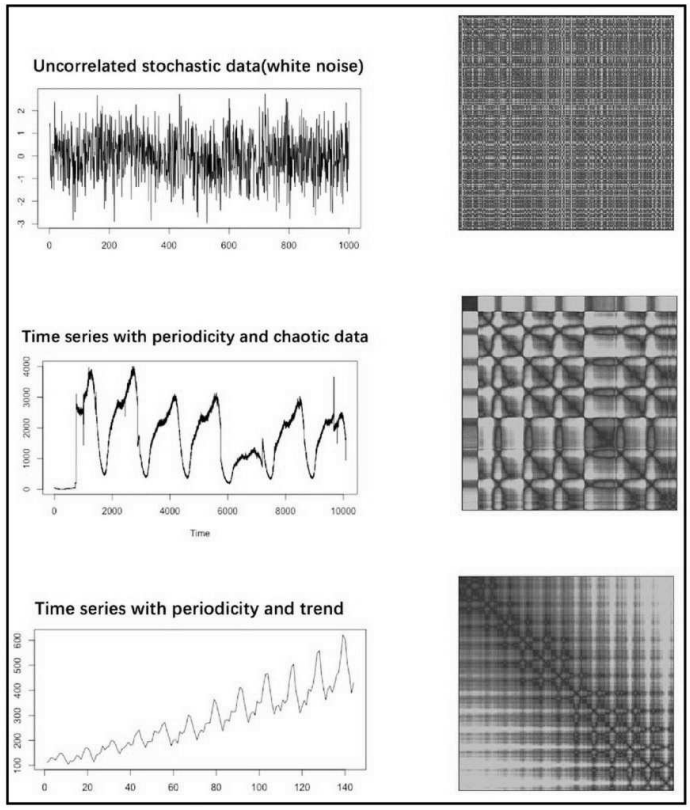

Figure 3.12: Typical examples of RPs for different time series.

small, and only single points or short lines can be discovered. The faster the divergence, i.e., the greater the Lyapunov exponent, the shorter the diagonals. The length of the lines parallel to the RP's principal diagonal reveals how quickly the trajectories diverge in phase space.

On the other side, isolated single points in an RP can appear when states are recurrent but infrequent or if they vary significantly or do not endure for any period of time, pointing towards a stochastic process. However, they are not the only indication of randomness or noise. Finally, when data changes slowly, square-like structures appear (sojourn points). Some typical examples of RPs for different time series are shown in Figure 3.12.

3.4 MARKOV CHAINS

Markov chains are a useful mathematical tool in stochastic systems. The fundamental idea is known as the Markov Property, which states that some stochastic process predictions may be made more straightforwardly by treating the future as independent of the past in light of the process's current state. This is used to make future state forecasts for stochastic processes easier to understand [145, 146].

Markov chains are utilized in a wide range of contexts because they can be designed to model many real-world processes. These disciplines include voice recognition, search engine algorithms, network security, mapping of animal population distributions, and more. Predicting asset and option values and estimating credit risks are two other applications [147].

The fundamental definitions needed to comprehend Markov chains are provided in the following sections. In addition, specific fundamental Markov chain characteristics and the particular example of finite state space Markov chains are investigated [148].

3.4.1 *Random Variables and Random Processes*

Before investigating Markov chains, let us go over some fundamental yet significant concepts in probability theory.

First, in mathematics, a random variable X is a variable whose value is determined by a random phenomenon. This result can be a number (or anything similar to a number, such as vectors) or something else. For illustration, we may define a random variable as the result of tossing a coin (not a number unless you assign, for instance, 0 to head and 1 to tail) or the outcome of rolling a dice (number). Also, keep in mind that the space of potential outcomes for a random variable might be discrete or continuous. For instance, a Poisson random variable is discrete, while a normal random variable is continuous.

Then, we can define a stochastic process (also known as the random process) as a set of random variables indexed by a set T that frequently denotes various instants in time (we will assume that in the following). Either T is the set of natural numbers

(discrete time random process) or T is the set of real numbers (continuous time random process) are the two most frequent scenarios. For instance, tossing a coin every day falls under the definition of a discrete-time random process. However, a continually fluctuating stock market option's price falls under the definition of a continuous-time random process. Random variables at different time instants can be dependent on each other in some way (for example, stock price) or independent of each other (for example, coin flipping), and they can have a continuous or discrete state space (space of possible outcomes at each instant of time).

3.4.2 *Markov Property and Markov Chain*

There are several well-known families of random processes, including the Poisson, Gaussian, Autoregressive, Moving-Average, Markov, and others. These particular cases have unique characteristics that let us study and comprehend them more thoroughly.

The Markov property is one that significantly simplifies the analysis of a random process. The Markov property states that, for a random process, if we know the value the process has taken at a particular moment, we will not be able to infer any further information about the behavior of the process in the future by understanding more about its previous behavior. More mathematically, for any assumed time, the future states' conditional distribution of the process given current and previous states depends just on the present state and not at all on the past states (the property known as memoryless). The term Markov process refers to a random process with the Markov property.

Now that the initial concept has been established, we are able to define homogeneous discrete-time Markov chains. Indeed, A Markov chain is a discrete state space and discrete-time Markov process. Therefore, a Markov chain is a discrete succession of states that satisfy the Markov property and are selected from a discrete state space (finite or infinite). A Markov chain can be represented mathematically as:

$$\mathbf{X} = (\mathbf{X}_n)_{n\in\mathbb{N}} = (\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, ...), \tag{3.13}$$

in which the process gets its values in a discrete set E at each instant of time, which signifies $\forall n \in \mathbb{N}$, $\mathbf{X}_n \in E$. The Markov property thus implies:

$$\begin{aligned}&\mathbb{P}(\mathbf{X}_{n+1} = s_{n+1} | \mathbf{X}_n = s_n, \mathbf{X}_{n-1} = s_{n-1}, \mathbf{X}_{n-2} = s_{n-2}, ...) = \\ &\mathbb{P}(\mathbf{X}_{n+1} = s_{n+1} | \mathbf{X}_n = s_n).\end{aligned} \tag{3.14}$$

Again, this last equation emphasizes the notion that, for a given period, the probability distribution for the future state is determined only by the present state and not by previous states.

3.4.3 *Random Dynamic of Markov Chains*

The preceding subsection established a broad framework that may be matched by any Markov chain. This subsection will illustrate the Markov chain's random dynamic.

Indeed, it can be difficult, if not impossible, to characterize a discrete-time random process that does not satisfy the Markov property. The probability distribution at a particular time can be determined by one or more time instants in the past and (or) future. Any accurate description of the process might be challenging because of all these potential temporal dependencies.

On the other hand, the dynamic of a Markov chain is easily defined thanks to the Markov property. Indeed, only two things must be specified: an initial probability distribution q_0 (i.e., a probability distribution at the time instant $n = 0$) represented by:

$$\mathbb{P}(\mathbf{X}_0 = s) = q_0(s), \qquad \forall s \in E, \tag{3.15}$$

and a transition probability kernel p (which indicates the probability that a state at time $n + 1$ would succeed to another at time n for each pair of states) represented by:

$$\mathbb{P}(\mathbf{X}_{n+1} = s_{n+1} | \mathbf{X}_n = s_n) = p(s_n, s_{n+1}), \forall (s_n, s_{n+1}) \in E \times E \tag{3.16}$$

With the previous two items specified, the entire (probabilistic) dynamic of the system is well-defined. Indeed, the probability of any realization of the process may thus be estimated in a recurrent manner.

Since they properly characterize the probabilistic dynamic of the process, many additional more complicated events may then be estimated using only the transition probability kernel p and the initial probability distribution q_0. One last fundamental relationship that should be mentioned is the statement of the probability distribution at time $n+1$ with respect to the probability distribution at time n:

$$\begin{aligned} q_{n+1}(s_{n+1}) &= \mathbb{P}(X_{n+1} = s_{n+1}) \\ &= \sum_{s \in E} \mathbb{P}(x_n = s)\mathbb{P}(x_{n+1} = s_{n+1} | x_n = s) \\ &= \sum_{s \in E} q_n(s)p(s, s_{n+1}). \end{aligned} \tag{3.17}$$

3.4.4 *Markov Chains Properties*

Some fundamental Markov chain characteristics or properties are provided in this subsection. The goal is not to delve deeply into the mathematical details but rather to provide a broad overview of the topics of interest that should be investigated while utilizing Markov chains. A Markov chain can be seen as a graph. The following will use the graphical form of Make chains to illustrate some of the characteristics. It should be noted, however, that these characteristics are not necessarily restricted to the finite state space scenario.

3.4.4.1 *Reducibility, Periodicity, Transience, and Recurrence*

If each state can be reached from any other state, even not in a single time step, the Markov chain is said to be irreducible. If the state space is finite and the Markov chain can be described by a graph, the graph that represents an irreducible Markov chain is referred to be strongly connected (Figure 3.13).

If any return to a state after leaving it needs multiple k time steps (k is the greatest common divisor of all available return route lengths), then the state has a period of k (Figure 3.14). A state is called aperiodic if $k = 1$, and a Markov chain is considered aperiodic if all of its states are aperiodic. If one state in an irreducible Markov chain is aperiodic, then all states in the chain are aperiodic.

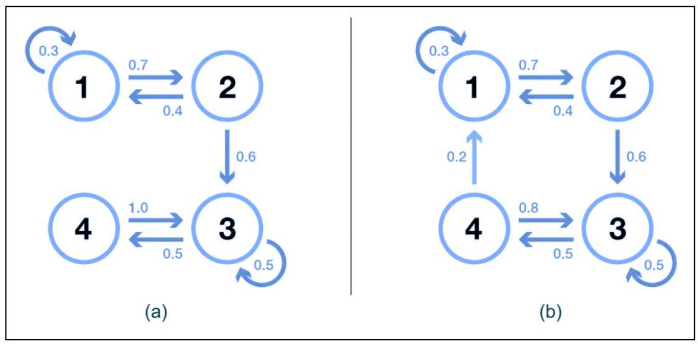

Figure 3.13: The illustration of the irreducibility property. (a) is not irreducible, and (b) is irreducible.

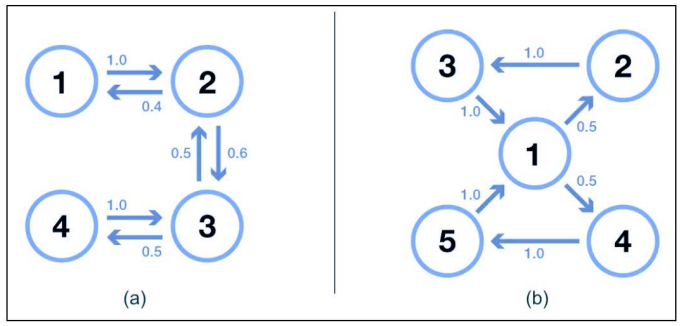

Figure 3.14: The illustration of the periodicity property. (a) is 2-periodic, and (b) is 3-periodic.

If there is a non-zero probability that we would never return to a state, it is said to be transient. In contrast, a state is said to be recurrent if we have a 1 probability of returning to it in the future after leaving it (Figure 3.15).

It is possible to determine the mean recurrence time for a recurrence state, which is the anticipated return time after leaving the state. Noticing that the estimated return time is not always finite even if the probability of return is equal to 1. Therefore, for the recurrent states, we can differentiate between the null recurrent state (infinite anticipated return time) and the positive recurrent state (finite anticipated return time).

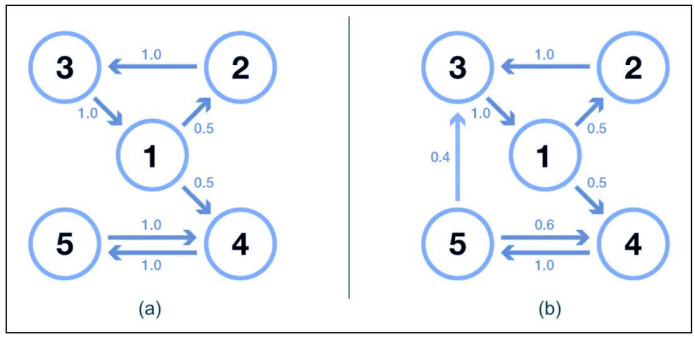

Figure 3.15: The illustration of the transience and recurrence property. (a) 1, 2, and 3 are transient, whereas 4 and 5 are recurrent. (b) is a full chain recurrent.

3.4.4.2 *Stationary distribution, limiting behavior, and ergodicity*

In this subsection, properties that characterize some aspects of the (random) dynamic described by a Markov chain are discussed.

A probability distribution like π is called a stationary distribution over the state space E if it satisfies the following equation:

$$\pi(e^{'}) = \sum_{e \in E} \pi(e) p(e, e^{'}), \qquad \forall e^{'} \in E. \tag{3.18}$$

Since the probability of being in $e^{'}$ at the current step is$\pi(e^{'})$ and the probability of being in $e^{'}$ at the next step is $\sum_{e \in E} \pi(e) p(e, e^{'})$, a stationary distribution confirms that the probability of being in $e^{'}$ at the current step is identical to the probability of being in $e^{'}$ at the next step.

Therefore, concerning the definition, a stationary probability distribution is one that does not change over time. Thus, the initial distribution q will remain the same for all subsequent time steps if it is a stationary distribution. If the state space is finite, p and π can be represented by a matrix and a raw vector, respectively, therefore:

$$\pi = \pi e = \pi e^2 = ... \tag{3.19}$$

It is important to note that an irreducible Markov chain only has a stationary probability distribution if all of its states are positive recurrent.

The following is another intriguing characteristic of stationary probability distributions. No matter what the initial probabilities are, if the chain is recurrent positive (therefore, there exists a stationary distribution) and aperiodic, then when time steps approach infinity, the probability distribution of the chain converges. In this case, the chain is considered to have a limiting distribution, which is just the stationary distribution. Generally, it can be expressed as, $\forall(e, e^{'}) \in E \times E$:

$$\lim_{n\to\infty} \mathbb{P}(\mathbf{X}_n = e^{'} | \mathbf{X}_0 = e) = \lim_{n\to\infty} p^n(e, e^{'}) = \pi(e^{'}). \tag{3.20}$$

It should be mentioned once again that there is no presumption on the initial probability distribution; regardless of the initial configuration, the probability distribution of the chain converges to the stationary distribution (equilibrium distribution of the chain).

The last intriguing characteristic of a Markov chain's behavior is ergodicity. If an irreducible Markov chain validates the subsequent ergodic theorem, it is said to as ergodic. Suppose that there exists an application $f(.)$ which maps the state space E to the real line (for example, it can be the cost to be in each state). For the n first terms, the mean value that this application takes along a particular trajectory (temporal mean) is represented by:

$$\frac{1}{n}(f(\mathbf{X}_0, \mathbf{X}_1, ..., \mathbf{X}_{n-1})) = \frac{1}{n}\sum_{i=0}^{n-1} f(\mathbf{X}_i). \tag{3.21}$$

The mean value of application f weighted by the stationary distribution (spatial mean) is computed as follows:

$$\sum_{e\in E} \pi(e) f(e). \tag{3.22}$$

Then, the ergodic theorem asserts that the spatial mean is identical to the temporal mean when the trajectory grows indefinitely long (weighted by stationary distribution). The ergodic property can be expressed as follows:

$$\frac{1}{n}\sum_{i=0}^{n-1} f(\mathbf{X}_i) = \sum_{e\in E} \pi(e) f(e). \tag{3.23}$$

In other words, at the limit, the early behavior of the trajectory becomes insignificant, and only the long-run stationary behavior is essential for computing the temporal mean.

4

RECURRENCE PLOTS-BASED ATTACK CLASSIFICATION

The advent of the Internet of Things (IoT), with the consequent changes in network architectures and communication dynamics, has affected the security market by introducing further complexity in traffic flow analysis, classification, and detection activities. Consequently, to face these emerging challenges, new empowered strategies are needed to effectively spot anomalous events within legitimate traffic and guarantee the success of early alerting facilities. However, such detection and classification strategies strongly depend on the right choice of employed features, which can be mined from individual or aggregated observations. Therefore, this work explores the theory of dynamic non-linear systems for effectively capturing and understanding the more expressive Internet traffic dynamics arranged as Recurrence Plots. To accomplish this, it leverages the abilities of Convolutional Autoencoders to derive meaningful features from the constructed plots. The achieved results, derived from a real dataset, demonstrate the effectiveness of the presented approach by also outperforming state-of-the-art classifiers. The main content of this chapter is based on one of our papers entitled 'Recurrence Plots-based Anomaly Detection using CNN-Autoencoders' [10].

4.1 INTRODUCTION

The success of IoT technologies, being the origin of significant changes in communication dynamics (e.g., affecting network layout/architecture, protocols, and interaction patterns), has strongly influenced the nature of Internet traffic introducing new security challenges.

In particular, due to the exponential growth of the number of IoT traffic sources and consequent traffic volumes (in terms of a superposition of individual flows on aggregation nodes and interfaces), distinguishing network anomalies became a difficult

task, in which randomness and background noise effects are more and more dominant. Also, the evolution from centralized or cloud-based solutions to edge service architectures, with the consequent flattening of infrastructure topologies and the security perimeter dissolution, introduced further complexity in traffic flow collection and interpretation.

Such interpretation, implying the understanding of the whole spectrum of the underlying traffic dynamics, requires careful analysis and correlation of the most characterizing traffic features with the final goal of discovering less evident (or often almost hidden) relationships between dynamics that describe the nature of a specific flow (i.e., a legitimate activity or a malicious one). Moreover, such a degree of complexity in the network traffic behavior cannot be effectively described with traditional traffic models and it has been shown [149] that, given the well-known chaotic nature of Internet traffic, the theory of nonlinear dynamical systems can be very helpful in providing extremely deep insights into traffic flow organization properties by highlighting the existence of periodic structures and recurrence phenomena [150] that are not evident at a glance and are significantly effective in discriminating events that deviate from normality. This is also due to the capability of nonlinear analysis of exploring the system's behavior in the phase space, which is a dimensionally richer representation of the system depicting its evolution pattern simultaneously on multiple time scales.

That means, in more detail, reconstructing and studying the system's attractor to appreciate the structure of multi-dimensional curves, also known as trajectories, formed in such space, corresponding to the system's evolution (or motion) over time. Such attractor is highly descriptive of the most significant, intrinsic, and discriminative system dynamics so that it can be used as an invaluable source of features for attack detection and classification.

A Recurrence Plot (RP) [49] is an extremely effective way of representing a system's behavior in phase space and hence visualizing its attractor as a two-dimensional image. Such image is characterized by large-scale, or typological patterns that can be homogeneous, periodic, drift, and disrupted as well as by the presence of small-scale structures, defining the specific texture,

identifying single dots, diagonal lines, vertical/horizontal lines, or bowed ones [36]. So, the structure of an RP is an effective tool for representing the behavior of a network flow, and hence for its classification as normal or anomalous [12].

Unfortunately, while being an extremely effective way of representing the system behavior, the visual interpretation of RPs is extremely difficult and requires a lot of experience. Accordingly, in this chapter, we exploit the potentialities of a Convolutional Sparse Autoencoders (CNN-SAE)-based neural network in extracting relevant spatial features from the RPs associated with traffic flows. To accomplish this, we start from some elementary statistical features (referred to as basic features) derived by sampling multiple consecutive observations over time and, thus, aggregated within the same temporal window. Then, by leveraging the aforementioned non-linear analysis framework we arrange them as multiple RPs (one for each basic feature), and hence multi-channel images. After that, a CNN-SAE is employed to derive new spatial features capable of capturing more complex and discriminating dynamics and thus significantly improving the DNNs-based classification process. Therefore, we employ a pipelined architecture by combining the following three main aspects, namely: extracting non-linear characteristics from the basic ones, finding out relevant features from the related RPs processed through the usage of a CNN-SAE, and then performing attack classification tasks by using the classification abilities of a fully-connected softmax DNN, respectively.

In addition, a comprehensive mathematical explanation of the overall workflow is provided to clarify how the proposed approach improves and affects the feature extraction process. Moreover, its effectiveness is investigated in presence of several unbalanced dataset partitions and finally compared with the most common and widely available state-of-the-art approaches.

Hence, the main contributions of this chapter can be summarized as follows [10]:

1. A formal description of the overall workflow is provided to investigate how the proposed approach, from the theoretical point of view, improves and affects the feature extraction process, respectively;

2. Non-linear characteristics, arranged as RPs, and the related spatial features extracted using a CNN-SAE are combined to perform attack classification tasks.

The remainder of this chapter is organized as follows. Section 4.2 will describe the proposed approach by providing a detailed mathematical formalization of the RPs and AEs. Section 4.3 will report the experimental phase and achieved results.

4.2 THE ATTACK CLASSIFICATION STRATEGY

This section presents the proposed approach for implementing the proposed attack classification strategy capable of exploiting the effectiveness of RPs in capturing traffic flow information. We accomplish this through three main steps, namely: determining new non-linear features from the basic ones (that is, building the RPs), extracting the more representative features from the RPs, and then performing classification using a Stacked Neural Network. More precisely, as shown in Figure 4.1, the RPs are built through the traditional delay-coordinate embedding process whereas the intermediate feature extraction step is performed by employing a CNN-SAE trained on the derived RPs. Then, for implementing the final classification step, we use the AE-related latent space as input to feed a fully-connected softmax neural network.

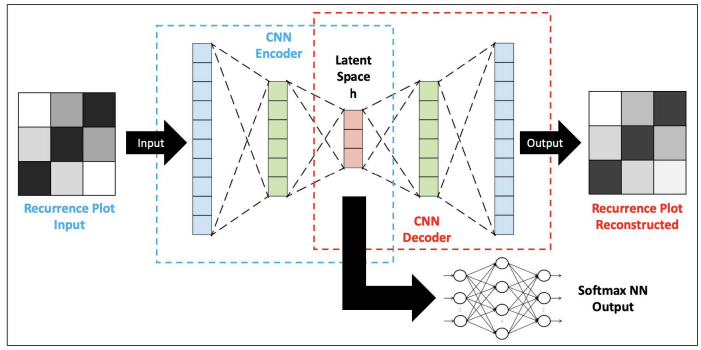

Figure 4.1: The proposed workflow.

As depicted in Figure 4.1, in this way, we combine the effectiveness of RPs in capturing traffic dynamics, the capability of AEs in

finding relevant features, and the classification abilities of DNNs. Therefore, in order to better present the proposed workflow's potentialities, we give a detailed description of each step by also providing a mathematical formulation of the involved CNN-SAE to explain how it is able to interpret RPs by determining new spatial features that differ from the well-known large-scale or small-scale structures commonly used in human-driven visual inspection.

4.2.1 *Determining recurrence plot-based features*

Understanding the hidden criteria and laws that rule network traffic behavior in certain moments is a very complex task, essentially due to the well-known irregularities and chaotic evolution trends that characterize its evolution. Such complexity is not easily manageable with the traditional event modeling arsenal and can be much better handled by using different concepts, abstractions, and points of view coming from the nonlinear analysis and dynamical system modeling framework.

Starting from these ideas, we can consider a system status $s_i(t)$ at time t as resulting from the superposition of a certain number of traffic flows, each described by specific features of interest $f_1, \ldots, f_n$, that are traversing the interface under observation at that time. Such a status evolves over time into another status $s_j(t + \Delta t)$ according to specific dynamics occurring in the time interval Δt, which can be referred to as normal activities or to an anomalous situation such as an attack. The entire evolution pattern characterizing the known system lifetime is described by the corresponding attractor, a geometrical abstraction describing the asymptotical evolution of the system status, whose study in certain points can reveal very complex properties providing deep insight into flow aggregation and clustering dynamics and revealing the presence of recurring structures that are not immediately evident at a glance and that can be used to really describe the behavior associated to normal or anomalous activities.

In order to better understand traffic dynamics under different conditions we specifically search for redundancies in the observed flows, in terms of a periodic recurrence of specific end-to-end communication patterns, inside traffic data. This is quite

straightforward in typical deterministic systems (e.g. describing an oscillatory phenomenon) where a return to a previously visited state occurs with a specific periodicity. Unfortunately, Internet traffic dynamics are known to be characterizable by using a chaotic system model, which, while deterministic, is aperiodic. In this case, it is not easy to discern temporal dependencies, but considering approximate repetitions of specific events we can build more complex rules that are able to give a deeper and better representation of the observed behavior. That is, we can still exploit recurrence phenomena to describe the system behavior if we consider the concept of recurrence within a specific threshold, and more specifically, so that we can consider a system state behaving as recurrent when it repeats its behavior after a certain quantity of time, also if not exactly, in a sufficiently close way.

As previously seen, RPs are an invaluable source of information for understanding the most representative properties of traffic flows. However, their interpretation is extremely complex and prone to errors, since it implies a high degree of subjectivity and expertise. Indeed, RPs contain a lot of interesting information about the system of interest concealed in subtle patterns that are not easily ascertained by visual inspection [12]. To overcome this issue, Zbilut and Webber [151] introduced the concept of Recurrence Quantification Analysis (RQA), which is an efficient and deterministic way of non-stationarity features identification in traffic flows. The core of such analysis is that it uncovers time correlations between data that are not based on linear or non-linear assumptions and cannot be distinguished through the direct study of one-dimensional series of traffic flow volumes [7, 152]. However, RQA is able to quantify only an extremely limited part of the information available in an RP, and more effective ways of extracting highly discriminant spatial features from such information are needed.

Therefore, in this chapter, we employ RPs images to gain significant insights related to network traffic flows by skipping the threshold-related value section issue using a gray scale for distances, in which white and black colors are used to represent short and long distances, respectively. Because network anomalies usually take place within temporal regions of a certain size, they cannot be spotted starting from individual traffic samples.

That is, individual observations occurring within a time series can not effectively capture anomalous events if picked one by one or isolated in a completely different context. Instead, the behavior that defines univocally an attack (and consequently the related features) becomes evident only when multiple temporal regions are considered together.

Hence, we do not scrutinize individual traffic flows as a sample at a specific time but consider aggregates of contiguous samples determined through the use of a sliding window scheme, better capturing the traffic behavior over time. This scheme is based on two distinct values: the sliding window size and the related. The offset parameter represents the time distance between two windows. The construction of RPs is performed through delay-coordinate embedding on independent windows of samples of a specific size, taken on the time series resulting from each basic feature, as shown in Figure 4.2. The time delay τ and the embedding dimension m used for such purpose have been determined at the training time on the entire set of flows constituting the training set, respectively as the first minimum of average mutual information function [153], and the embedding dimension m, by using the False Nearest Neighbors (FNN) method [143].

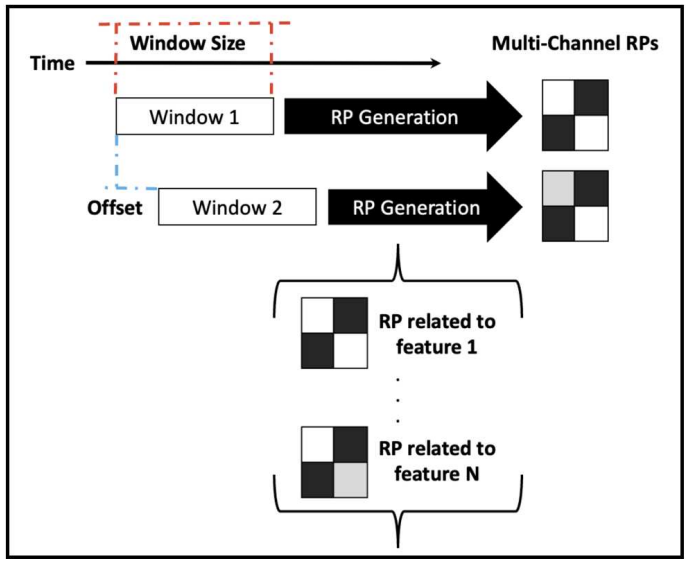

Figure 4.2: Schema of the sliding windows for RPs aggregation.

The window size has to be long enough to avoid the well-known curse of dimensionality phenomenon, making all the obtained estimations essentially meaningless. On the other hand, it must not be too big to contain the classification latency. Of course, it also depends on the sampling interval to guarantee the availability of a sufficient number of observations falling within a window [149].

Also, we aggregated RPs associated with the different basic features available in sample observations as multi-channel images, in which each channel represents one of the employed features, by improving their descriptive power.

4.2.2 *Using CNN Autoencoders for extracting spatial features from RPs*

In this configuration, we suppose that the AE used for processing aggregated RPs, with reference to Figure 4.1, includes only three layers, namely: input, hidden, and output, respectively.

Let $x \in \mathbb{R}^d$ be the input vector of Figure 4.1, and let $x^{AE} \in \mathbb{R}^{d'}$ be the input of the considered AE, then $x \equiv x^{AE}$ and $d = d'$.

As explained in Section 3.2.4, an AE seeks to reconstruct the input by encoding it to a latent space h, which is then decoded to an output $\tilde{x}^{AE}$ defined as follows:

$$\tilde{x}^{AE} = y_{(W',b')}(h_{(W,b)}(x^{AE})) \equiv x^{AE} \tag{4.1}$$

where (W, b) and (W', b') represent the matrix of the weights and the bias vector of the encoder and decoder respectively, whereas y is the decoder activation function.

Moreover. let n be the number of hidden neurons of AE, then $W \in \mathbb{R}^{n \times d'}$, $b \in \mathbb{R}^n$, and $h_{(W,b)}$ is given by:

$$h_{(W,b)}(x^{AE}) = \sigma(Wx^{AE} + b) \tag{4.2}$$

where σ is the encoder activation function.

Since an Autoencoder is trained by minimizing a loss function $\mathcal{F}$, it is possible to consider additional constraints, also referred to as regularization terms, to give the AE some specific capabilities.

For instance, Sparse AEs are often employed to extract meaningful and relevant features from input data and, consequently, to improve the classification results. More precisely, Sparsity can be obtained through different strategies (e.g., L1 regularization and KL regularization) by forcing the involved AE to have only a few simultaneously active nodes (1 in theory) that, as a result, positively affects the learning process [154]. Concerning SAEs, their regularization is accomplished by adding a penalty term to the loss function, that is:

$$\mathcal{F}(x^{AE}, \tilde{x}^{AE})_{Sparse} = \mathcal{F}(x^{AE}, \tilde{x}^{AE}) + \lambda \mathcal{S}(W, b) \tag{4.3}$$

where λ expresses the degree of regularization, and $\mathcal{S}(W, b)$ represents the sparsity-related term.

Once the training process is completed, the output of the l^{th} hidden neuron h_l can be derived by:

$$h_l = \sigma(\sum_{k=1}^{d'} w_{lk} x_k^{AE} + b_l) \tag{4.4}$$

Hence, since the input data of a Sparse AE is constrained by $||x^{AE}||^2 \leq 1$, each input data component x_k^{AE} activating the l^{th} neuron is given by:

$$x_k^{AE} = \frac{w_{lk}}{\sum_{m=1}^{d'} (w_{lm})^2}, \ \forall \ k, m = 1 \ ... \ d'. \tag{4.5}$$

which extracts a feature exactly corresponding to the l^{th} output node. That means that a Sparse AE can learn different sets of characteristics from input data at least equal to the number of considered hidden neurons n.

As described in Section 3.2.5, AEs are frequently coupled with different DNNs flavors to add new functionalities and improve the ability to mine more complex features from input data. Thus, to fully exploit the effectiveness of non-linear characteristics, we use several Convolutional layers to derive relevant relations from the related aggregated RPs that can be considered as spatial features [54].

Let $X \in \mathbb{R}^{N_x \times N_y}$ be the CNN-related input, $RP \in \mathbb{R}^{N \times N}$ be an aggregated Recurrence Plot of Figure 4.2, and $C^f \in \mathbb{R}^{a \times b}$ be the f^{th} filter, respectively. Then, $X \equiv RP$ and $N_x \times N_y = d = N \times N$.

On these assumptions, the convolution operation, applied on the CNN-input RP with N_f filters, is defined by:

$$F_{i,j} = \sum_{f=1}^{N_f} \sum_{p=1}^{a} \sum_{q=1}^{b} C^f_{p,q} RP_{i+p-1,j+q-1} \tag{4.6}$$

with $F_{i,j}$ the components of the filtered input F.

The size of F is defined through its row F_x and column F_y dimensions, by:

$$\begin{aligned} F_x &= \frac{N_x - a + 2P}{S_x} + 1 \\ F_y &= \frac{N_y - b + 2P}{S_y} + 1 \end{aligned} \tag{4.7}$$

where P is the Padding referring to the number of zeros around the border of X, while S_x and S_y are the Strides related to the row and column, which control the shifting of the filter on the input matrix.

Since $d = N_x \times N_y$, it is possible to map any x_k to a point $RP_{i,j}$ into a two-dimensional array. Thus, with abuse of notation, we can assert that:

$$x_k \equiv RP_{i,j} \tag{4.8}$$

with $k = 1, ..., d$, $i = 1, ..., N_x$, and $j = 1, ..., N_y$.

By substituting $F_{i,j}$ of Equation 4.6 into x_k^{AE} of Equation 4.4, it yields:

$$h_l = \sigma(\sum_{k=1}^{d'} w'_{lk} x_{\phi(k)} + b_l) \tag{4.9}$$

such that $d' = F_x \times F_y$, while

$$w'_{lk} = \sum_{f=1}^{N_f} \sum_{p=1}^{a} \sum_{q=1}^{b} C^f_{p,q} w_{lk} \tag{4.10}$$

and

$$x_{\phi(k)} \equiv RP_{i+p-1,j+q-1} \tag{4.11}$$

Analogously to Equation 4.4 and Equation 4.5, Equation 4.9 and Equation 4.10 indicate that w'_{lk} (for each l, k) represent the new extracted features that express a more complex knowledge because they are defined as the linear combination of the original w_{lk}. Therefore, the employed CNN-SAE configuration can process non-linear characteristics, arranged as RPs, by providing meaningful features to feed the fully-connected softmax neural network.

4.3 PERFORMANCE EVALUATION AND RESULTS ANALYSIS

The presented experiments and their results are devoted to demonstrating the effectiveness of the proposed approach leveraging RPs for representing network traffic anomalies as multi-channel images and automatically interpreting them through the extraction of highly discriminative spatial classification features. To accomplish this, we employ the learning abilities of CNN Autoencoders trained on a famous dataset, which includes packets generated by different workstations, protocols, and applications.

4.3.1 *Dataset and Basic Features*

For the following experiments, we used real-world traffic features available in the Intrusion Detection Evaluation Dataset (CIC-IDS2017)[1] [155], which are composed of several .csv files related to Benign and Attack activities. More precisely, we considered all the Normal traffic captured during Monday 03/07/2017 and Malicious traffic captured from Tuesday 04/07/2017 to Friday 07/07/2017. Table 4.1 summarizes the traffic nature and related activities for each day and each employed .csv file, respectively.

Since the used dataset provides more than 80 network-related features, we pre-processed it in order to represent each bidirectional flow (summarized as a single dataset entry) as a low-

1 https://www.unb.ca/cic/datasets/ids-2017.html

Day/CSV Name	Typology	Description
Monday	Normal	Normal human activities
Tuesday	Attack	Brute Force (FTP/SSH-Patator)
Wednesday	Attack	DoS/DDoS
Thursday	Attack	Brute Force (FTP/SSH-Patator)
Friday1	Attack	Port Scan
Friday2	Attack	DDoS

Table 4.1: Overview on the employed .csv files.

dimensional vector characterized by only the following nine basic features:

- *Total Fwd Packets*: number of sent packets from Sender;
- *Total Bwd Packets*: number of sent packets from Receiver;
- *Flow Bytes/s*: flow of bytes for second;
- *Flow Packets/s*: flow of exchanged packets for second;
- *Average Packet Size*: average number of sent packets;
- *Packet Length Mean*: mean length of the exchanged packet;
- *Down/Up Ration*: the Down/Up ratio estimated for the considered flow;
- *SYN Flag Count*: number of set SYN Flags;
- *ACK Flag Count*: number of set ACK Flags.

Then, we used the related ground-truth to associate each flow with its corresponding typology (normal or each type of attack), thus adding the supervisory signal. Finally, since the basic features had characterized by different scales, we considered several rescaling approaches that have been applied to the whole dataset to remove any possible bias situations. Accordingly, following the achieved results, we rescaled each .csv file employing a standard normal distribution, with zero mean and standard deviation equal to 1 ($\mu = 0$, $\sigma = 1$).

4.3.2 *Experimental setting*

Once the basic features have been selected and scaled, we merged all related .csv files as a whole dataset in order to estimate the embedding dimension m and the time_delay τ. More precisely, for each feature-related column, we performed the False Nearest Neighbors (FNN) Algorithm [143] and derived the Average Mutual Information (AMI) using the Non-Linear Time Series Analysis (NoLiTSA) Python framework [156]. Therefore, we set the minimum value $m = 3$ and $\tau = 1$ because they were the common values capable of minimizing the false neighbors and the AMI, respectively. Then, we split the employed dataset into two mutually-exclusive sets using, for each .csv file, the 70% of vectors for training and the remaining 30% for testing. The resulting .csv files were aggregated to form the overall training and testing datasets, respectively. Note that the temporal order was conserved during such a splitting process. Next, by following the schema proposed in Figure 4.2, several sliding windows characterized by having different dimensions and offsets have been considered, as reported below:

- winDim: dimension for each window (8, 16, 32, 64, 128);
- offset: distance between two windows (1 - Consecutive, winDim/2 - semi-overlapped, winDim - non-overlapped).

More precisely, for each set of flows (each characterized by the 9 basic features mentioned above) falling within a specific sliding window, the related RPs as $(winDim - (m-1)\tau) \times (winDim - (m-1)\tau)$ matrices for each considered basic feature were first generated. Then, the resulting RPs (one for each basic feature) were aggregated together to form a single multi-channel (9-channels) RP image, which characterizes the set of traffic flows falling within the considered time window. Furthermore, in order to classify a single traffic flow, each derived RP was labeled so as to assume the label of the last flow belonging to the considered time window. In such a way, each flow is associated with a specific RP and is evaluated by taking into account the previous flows (i.e. the traffic history).

Although many experiments were carried out by using different combinations of $winDim/offset$, $winDim = 16$ showed

to get best results. Therefore, in all experiments, we used this dimension for the time window. Table 4.2, Table 4.3, and Table 4.4 show the details of the training and testing sets derived by using a time window of size 16 and offset values 1, 8, and 16, respectively.

Category	Training	Testing	Total
DDoS	89601	38385	127986
DoS	172275	73795	246070
FTP-Patator	5537	2353	7890
Port Scan	111137	47617	158754
SSH-Patator	4097	1745	5842
Normal	370609	158817	529426
Total	753256	322712	1075968

Table 4.2: Dataset division with winDim = 16 and offset = 1.

Category	Training	Testing	Total
DDoS	11201	4799	16000
DoS	21537	9227	30764
FTP-Patator	693	295	988
Port Scan	13893	5953	19846
SSH-Patator	513	219	732
Normal	46327	19853	66180
Total	94164	40346	134510

Table 4.3: Dataset division with winDim = 16 and offset = 8.

Finally, the dataset pre-processing phase, the related Recurrence Plots extraction, and all the experiments have been conducted with an iMac Desktop equipped with an Intel 6-core *i*7 CPU @ 3.2 GHz and 16 GB RAM.

Category	Training	Testing	Total
DDoS	5601	2400	8001
DoS	10770	4615	15385
FTP-Patator	347	148	495
Port Scan	6947	2977	9924
SSH-Patator	257	110	367
Normal	23164	9927	33091
Total	47086	20177	67263

Table 4.4: Dataset division with winDim = 16 and offset = 16.

4.3.3 *Evaluation Metrics*

To assess the classification effectiveness of the presented scheme, we considered the traditional evaluation metrics that can be extracted from the confusion matrix: Accuracy (Acc.), Sensitivity (Sens.), Specificity (Spec.), Precision (Prec.), F-Measure (F-Mea.), and Area Under the ROC Curve (AUC), as defined below.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.12}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{4.13}$$

$$Specificity = \frac{TN}{TN + FP} \tag{4.14}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.15}$$

$$F - Measure = \frac{2 * Sens * Prec}{Sens + Prec} \tag{4.16}$$

$$AUC = \frac{Sens + Spec}{2} \tag{4.17}$$

Where for each category, TPs (True Positives) are the flows correctly classified, FPs (False Positives) are the flows incorrectly classified, FNs (False Negatives) are the flows incorrectly rejected, and TNs (True Negatives) are the flows correctly rejected.

4.3.4 *Model description and Results*

As mentioned earlier, we used a CNN-SAE for interpreting RPs. Accordingly, in this Section, the implementation details are first reported, and then the experimental results are shown.

Figure 4.3 shows the high-level organization of the CNN layers employed for the encoder side. As depicted, it includes a sequence of three Conv2D layers with kernel_size=$(2,2)$, activation=relu, and padding=same, characterized by having 18, 8, and 4 filters, respectively. After that, a Flatten layer is employed to map the extracted features as one-dimensional latent vectors. Hence, after having built the decoder side using the inverse sequence of the encoding layers, we trained the CNN-SAE configuration by using Adam optimizer as well as the Mean Absolute Error (MAE) as loss function for 50 epochs and batch_size = 64.

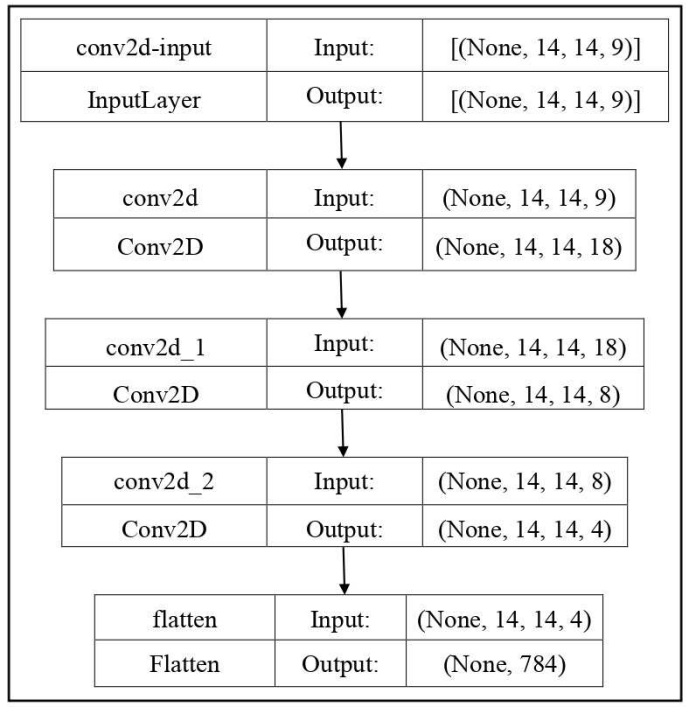

Figure 4.3: CNN layers high-level organization.

Next, we fed a fully-connected softmax neural network comprising two Dense layers with 512 neurons for each layer, activa-

tion=relu for both layers, and dropout=0.5, 0.4, respectively. Also, to obtain the classification results as probability distributions, we used a third dense layer with 6 neurons and activation=softmax as the output layer. Then, we trained the whole network with Adam optimizer and SparseCategoricalFocalLoss loss function for 20 epochs and batch_size = 256.

Many experiments were carried out by varying the following hyper-parameters:

- numConvLayers: the number of Conv2D layers considered (1, 2, 3);
- numFilters: the number of filters considered for each Conv2D layer (1, 2, 4, 8, 12, 14, 16, 18, 28, 36);
- strides: the stride length and combinations for each Conv2D layer $((1,1), (1,2), (2,1), (2,2))$;
- numDenseLayers: the number of Dense layers considered (1, 2, 3, 4);
- numNeurons: the number of neurons considered for each Dense layer (16, 24, 28, 32, 64, 96, 128, 256, 512, 1024);
- dropout: dropout values for each Dense layer (0.05, 0.1, 0.2, 0.3, 0.4, 0.5);
- activation: activation functions employed (relu, softmax, sigmoid);
- batch_size: considered batch_size values (32, 64, 128, 256, 512, 1024, 2048);
- loss: loss functions used (Mean Squared Error (MSE), Mean Absolute Error (MAE), SparseCategoricalCrossentropy, SparseCategoricalFocalLoss).

Note that only the configurations that achieved the best results were reported in this Section.

Table 4.5, Table 4.6, and Table 4.7 report the derived Confusion Matrices, while Table 4.8, Table 4.9, and Table 4.10 summarize the corresponding classification metrics derived by fine-tuning the whole CNN-SAE-NN stacked network for 20 additional epochs.

	DDoS	DoS	FTP-Patator	Port Scan	SSH-Patator	Normal
DDoS	38149	215	0	14	0	7
DoS	1992	70734	0	661	0	408
FTP-Patator	0	0	2331	0	4	18
Port Scan	90	122	0	47303	0	102
SSH-Patator	0	0	0	0	1745	0
Normal	6	52	0	0	0	158759

Table 4.5: Confusion Matrix (offset = 1).

	DDoS	DoS	FTP-Patator	Port Scan	SSH-Patator	Normal
DDoS	4761	30	0	3	0	5
DoS	351	8712	0	97	0	67
FTP-Patator	0	0	291	0	0	4
Port Scan	2	13	0	5925	0	13
SSH-Patator	0	0	0	0	219	0
Normal	0	10	0	0	0	19843

Table 4.6: Confusion Matrix (offset = 8).

	DDoS	DoS	FTP-Patator	Port Scan	SSH-Patator	Normal
DDoS	2365	21	0	5	0	9
DoS	118	4251	0	103	1	142
FTP-Patator	0	0	145	0	0	3
Port Scan	3	5	0	2962	0	7
SSH-Patator	0	0	0	0	109	1
Normal	0	2	0	0	0	9925

Table 4.7: Confusion Matrix (offset = 16).

4.3.5 *Comparison and Discussion*

To better demonstrate the effectiveness of the proposed classification framework, we compare the obtained results with those derived by the most famous state-of-the-art ML-based classifiers provided by WEKA [157], such as Naive Bayes-based classifier (NB), Logistic classifier (LOG), Support Vector Machine classifier (SVM), J48 decision-tree classifier (J48), and Multilayer Perceptron classifier (MLP), specifically arranged for attack detection and classification. Moreover, in order to compare the proposed method with another model based on DL, we examined several

Category	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
DDoS	0.9928	0.9927	0.9481	0.9939	0.9704	0.9933
DoS	0.9893	0.9984	0.9945	0.9585	0.9762	0.9785
FTP-Patator	0.9999	1.0000	1.0000	0.9907	0.9953	0.9953
Port Scan	0.9969	0.9975	0.9859	0.9934	0.9897	0.9955
SSH-Patator	1.0000	1.0000	0.9977	1.0000	0.9989	1.0000
Normal	0.9982	0.9967	0.9966	0.9996	0.9981	0.9982
Avg.	0.9962	0.9976	0.9872	0.9893	0.9881	0.9935

Table 4.8: Statistic Metrics related to dataset with offset = 1.

Category	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
DDoS	0.9903	0.9901	0.9310	0.9921	0.9606	0.9911
DoS	0.9859	0.9983	0.9940	0.9442	0.9684	0.9712
FTP-Patator	0.9999	1.0000	1.0000	0.9864	0.9932	0.9932
Port Scan	0.9968	0.9971	0.9834	0.9953	0.9893	0.9962
SSH-Patator	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Normal	0.9975	0.9957	0.9955	0.9995	0.9975	0.9976
Avg.	0.9951	0.9969	0.9840	0.9863	0.9848	0.9916

Table 4.9: Statistic Metrics related to dataset with offset = 8.

Category	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
DDoS	0.9923	0.9932	0.9513	0.9854	0.9606	0.9893
DoS	0.9806	0.9982	0.9935	0.9211	0.9684	0.9597
FTP-Patator	0.9999	1.0000	1.0000	0.9898	0.9932	0.9899
Port Scan	0.9939	0.9937	0.9648	0.9797	0.9893	0.9943
SSH-Patator	0.9999	1.0000	0.9909	0.9909	1.0000	0.9954
Normal	0.9919	0.9842	0.9839	0.9998	0.9918	0.9920
Avg.	0.9931	0.9949	0.9807	0.9787	0.9794	0.9868

Table 4.10: Statistic Metrics related to dataset with offset = 16.

CNN and convolutional sparse autoencoder architectures that are very effective in image processing. In this section, we report

just the results achieved from the best architecture (in terms of performance metrics) [158].

Table 4.11 summarizes the performance metrics related to the employed CNN-SAE-NN configuration for each considered offset.

Offset	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
offset = 1	0.9962	0.9976	0.9872	0.9893	0.9881	0.9935
offset = 8	0.9951	0.9969	0.9840	0.9863	0.9848	0.9916
offset = 16	0.9931	0.9949	0.9807	0.9787	0.9794	0.9868

Table 4.11: Avg. Statistic Metrics comparison related to each considered offset value.

As shown in Table 4.11, the CNN-SAE configuration achieved the best classification results for each considered offset value (consecutive, semi-overlapped, and non-overlapped). More precisely, we have obtained an average F-Measure close to 99% for the consecutive scenario and more than 98% for the semi-overlapped scenario, while we obtained a worsening of only less than 1% for the non-overlapped case.

Finally, as mentioned above, we analyzed the effectiveness of our approach by comparing the achieved results with those derived from NB, LOG, SVM, J48, MLP, and CNN-based classifiers. More precisely, we have evaluated their classification capabilities in comparison with our method by considering non-overlapped time windows with offset = 16, which is the worst scenario of our approach (see table Table 4.10). Table 4.12, Table 4.13, Table 4.14, Table 4.15, Table 4.16, and Table 4.17 summarize the average metrics derived by the employed ML and DL-based approaches, while Table 4.18 reports the comparison between our proposed neural network configuration and these classifiers (only Avg. values are reported).

As shown in Table 4.18, the proposed CNN-SAE network drastically outperforms Naive Bayes-based, Logistic, Support Vector Machine, J48 decision-tree, Multilayer Perceptron, and Convolutional Neural Network-based classifiers by significantly improving the average of all evaluation metrics, specifically F-Measure. It proves, once again, the effectiveness of the proposed approach

Category	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
DDoS	0.7126	0.7286	0.2281	0.5940	0.3296	0.6613
DoS	0.8164	0.9967	0.9493	0.2086	0.3420	0.6026
FTP-Patator	0.9279	0.9313	0.0476	0.4624	0.0864	0.6969
Port Scan	0.9133	0.9001	0.6316	0.9895	0.7711	0.9448
SSH-Patator	0.6780	0.6762	0.0167	0.9977	0.0328	0.8370
Normal	0.5145	0.9967	0.8302	0.0165	0.0324	0.5066
Avg.	0.7604	0.8716	0.4506	0.5448	0.2657	0.7082

Table 4.12: Statistic Metrics related to NB classifier.

Category	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
DDoS	0.9607	0.9722	0.8097	0.8749	0.8410	0.9236
DoS	0.9170	0.9783	0.9067	0.7101	0.7965	0.8442
FTP-Patator	0.9910	0.9981	0.1392	0.0416	0.0640	0.5198
Port Scan	0.9902	0.9898	0.9440	0.9925	0.9676	0.9912
SSH-Patator	0.9939	0.9994	0.0000	0.0000	0.0000	0.4997
Normal	0.9415	0.9006	0.9055	0.9838	0.9430	0.9422
Avg.	0.9657	0.9731	0.6175	0.6005	0.6020	0.7868

Table 4.13: Statistic Metrics related to LOG classifier.

Category	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
DDoS	0.9154	0.9978	0.9496	0.3045	0.4612	0.6512
DoS	0.8819	0.9494	0.7933	0.6542	0.7171	0.8018
FTP-Patator	0.9926	1.0000	0.3333	0.0004	0.0008	0.5002
Port Scan	0.9853	0.9844	0.9164	0.9908	0.9522	0.9876
SSH-Patator	0.9924	0.9979	0.0015	0.0006	0.0008	0.4992
Normal	0.8538	0.7384	0.7826	0.9730	0.8675	0.8557
Avg.	0.9369	0.9446	0.6295	0.4873	0.4999	0.7160

Table 4.14: Statistic Metrics related to SVM classifier.

and the abilities of Autoencoders that, in any application scenario (e.g., dataset partitions), are capable of deriving meaningful fea-

Category	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
DDoS	0.9995	0.9999	0.9992	0.9964	0.9978	0.9982
DoS	0.9816	0.9970	0.9893	0.9295	0.9584	0.9632
FTP-Patator	0.9997	1.0000	1.0000	0.9580	0.9785	0.9790
Port Scan	0.9934	0.9925	0.9586	0.9985	0.9782	0.9955
SSH-Patator	0.9971	1.0000	1.0000	0.4797	0.6483	0.7398
Normal	0.9845	0.9737	0.9735	0.9956	0.9844	0.9847
Avg.	0.9926	0.9937	0.9868	0.8929	0.9243	0.9434

Table 4.15: Statistic Metrics related to J48 classifier.

Category	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
DDoS	0.9324	0.9992	0.9872	0.4373	0.6061	0.7183
DoS	0.9090	0.9720	0.8806	0.6968	0.7780	0.8344
FTP-Patator	0.9963	1.0000	0.9983	0.5027	0.6687	0.7514
Port Scan	0.9980	0.9993	0.9962	0.9904	0.9933	0.9949
SSH-Patator	0.9945	1.0000	0.0000	0.0000	0.0000	0.5000
Normal	0.8689	0.7490	0.7929	0.9928	0.8817	0.8709
Avg.	0.9499	0.9533	0.7759	0.6034	0.6546	0.7783

Table 4.16: Statistic Metrics related to MLP classifier.

Category	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
DDoS	0.9866	0.9877	0.9150	0.9783	0.9456	0.9830
DoS	0.9796	0.9975	0.9909	0.9194	0.9538	0.9584
FTP-Patator	0.9992	1.0000	1.0000	0.8919	0.9429	0.9459
Port Scan	0.9961	0.9970	0.9827	0.9913	0.9870	0.9941
SSH-Patator	0.9990	0.9999	0.9688	0.8455	0.9029	0.9227
Normal	0.9910	0.9828	0.9826	0.9995	0.9910	0.9912
Avg.	0.9919	0.9942	0.9733	0.9376	0.9539	0.9659

Table 4.17: Statistic Metrics related to CNN classifier.

tures from traffic dynamics arranged as RPs and thus achieve excellent classification results.

Method	Acc.	Spec.	Prec.	Sens.	F-Mea.	AUC
NB	0.7604	0.8716	0.4506	0.5448	0.2657	0.7082
LOG	0.9657	0.9731	0.6175	0.6005	0.6020	0.7868
SVM	0.9369	0.9446	0.6295	0.4873	0.4999	0.7160
J48	0.9926	0.9937	0.9868	0.8929	0.9243	0.9434
MLP	0.9499	0.9533	0.7759	0.6034	0.6546	0.7783
CNN	0.9919	0.9942	0.9733	0.9376	0.9539	0.9659
Proposed	0.9931	0.9949	0.9807	0.9787	0.9794	0.9868

Table 4.18: Comparison between the proposed configuration and ML and DL-based approaches (only Avg. values are reported).

5

MALWARE DETECTION USING FEDERATED MARKOV CHAINS

The continuous emergence of new and sophisticated malware specifically targeting Android-based Internet of Things devices is causing significant security hazards and is consequently fostering the need for effective detection models and strategies able to work with these hardware-constrained devices. In addition, since such models are often trained on confidential application data, many involved subjects are reluctant to share their data for this purpose. Accordingly, several Federated Learning-based solutions are emerging, which rely on the capabilities of Machine Learning models in malware detection/classification without sharing user data. However, Federated Learning methods are often adversely affected by non-independent and identically distributed data in terms of both the required training time and classification results. Therefore, a promising solution could be to overcome the Federated Learning-related issues by preserving the privacy of end-user data. In this direction, the capabilities of Markov chains and associative rules are extended within a federated environment to support malware classification tasks in the IoT scenario.

The presented approach in this chapter, evaluated on several malware families, has achieved an average accuracy of 99% in the presence of centralized and decentralized unbalanced training/testing data by overcoming the most common state-of-the-art approaches. Also, its runtime performance is comparable with centralized ones by considering several non-independent and identically distributed dataset partitions, splitting criteria, and clients, respectively. The main content of this chapter is based on one of our papers entitled 'Privacy-preserving Malware Detection in Android-based IoT Devices Through Federated Markov Chains' [11].

5.1 INTRODUCTION

The exponential growth of IoT technology together with the success of the Android platform caused the explosion of the number of mobile apps targeted for many flavors of Android-based IoT devices such as smart TVs, personal assistants, smart wearables, toasters, refrigerators, treadmills, and other interconnected electronic gadgets that can be easily controlled by using common smartphones, resulting in a major driving force within the IoT ecosystem. Also, Android-based smartphones are assuming very frequently the role of IoT gateways for many wearable or home/vehicle-based mission-critical IoT applications, involving e-health, domotics, assisted driving, etc. Unfortunately, despite the fundamental role of these architectures in the IoT forefront, they introduce new cybersecurity issues and risks [159]. In particular, due to the lack of appropriate security protection mechanisms on most of the simplest Android-based embedded platforms empowering IoT devices, the large volume of yearly-released malware applications specifically targeted for these environments poses challenges that foster the introduction of effective detection and classification techniques [160].

The extensive usage of concealment and obfuscation strategies is one of the primary success factors for the malware development market and a major cause for the significant growth of the known number of malware samples targeting IoT devices.

Therefore, to tackle the fast development and evolution of malware in the IoT environment, it is crucial to design robust and reliable malware detection and classification strategies that are sensitive and effective against different malware families [161]. According to earlier malware classification studies, malware samples usually belong to a family with similar behaviors, implying that most new malware is derived as new versions of already existing malware. Hence, the possibility of developing strategies that can effectively categorize malware depending on its family, regardless of being a variant, appears particularly promising to prevent and control its evolution over time.

Many different dynamic analysis-based techniques have been proposed, considering the dynamic behavior of malware applications through the observation of the frequencies and/or se-

quences of system-level Application Programming Interface (API) calls. These strategies are based on the idea that malicious applications might contain a set of well-distinct APIs invoked in a different order compared to those related to goodware applications [162]. Moreover, some particular API calls may be used significantly more often in malicious code fragments [60, 67, 163].

Markov chains are one of the most effective cutting-edge dynamic analysis-based strategies that can model the API calls invoked by malware applications and construct the representative behavioral patterns of particular malware families [8]. More precisely, they consider the sequence of API calls to model the application-related behavior as a graph where each node represents a unique API, while each edge represents the transition probability between two APIs. Markov chain-based detectors have also been proven resistant to evasion efforts carried out by selectively inserting irrelevant API calls throughout malicious code [9].

ML-based models represent another very successful approach for detecting malware threats. They can exploit large datasets to train the adopted models according to centralized and decentralized solutions. However, building a sufficiently complete knowledge base on emerging malware attacks is currently a slow and challenging process. In addition, the involved organizations (developers and/or end-users) are often unwilling to share their data since they are concerned about disclosing their intellectual property and sensitive data about their IoT applications and systems. Also, they are unable to keep up-to-date against new malware attacks and are constantly at a disadvantage against them.

Alternatively, Federated Learning (FL)-based approaches represent recent privacy-preserving solutions, leveraging ML and DL models' capabilities in several classification and detection tasks without sharing their data [6, 76, 78]. However, as highlighted in many literature studies, non-Independent and Identically Distributed (non-IID) data often adversely affects FL-based models regarding the required training time, convergence, learning processes, and classification results [20, 82–84]. Also, the proposed learning strategies are often strongly influenced by the configura-

tion of some additional hyperparameters (e.g. threshold values) which could limit their applicability.

Aimed by this motivation, in this chapter, we present a federated Markov chains-based paradigm for malware detection in Android-based IoT scenarios empowered by dynamic execution analysis with system-level API calls monitoring. Such paradigm makes data owners proactive contributors to the ML-based detector model building, giving them the means to timely update a global model without sharing their private raw data (and hence without disclosing their API execution history and/or applications installed). More precisely, Markov chains and associative rules are used within a federated logic, in which users independently process the raw API execution data of each application and then send the extracted information to a central server, primarily dedicated to setting up and sharing the detector in a way that protects user privacy. Next, we analyze the effectiveness of the proposed strategy, by comparing it with the most common state-of-the-art ML-based approaches, within a realistic IoT scenario in which we used a dataset of around 3500 malware belonging to 8 Android families. Finally, we evaluate the required time effort by considering several dataset partitions and involved clients. Therefore, we show that the proposed federated architecture can obtain comparable time performances in the presence of non-IID data.

The main contributions of this chapter can be summarized as follows [11]:

1. A federated architecture is presented to support the rules mining process as multiple Markov chains;

2. The resulting associative rules-based detector is exploited to recognize and classify several malware families by considering both centralized and decentralized data;

3. A performance study is done to show the effectiveness of the proposed approach in the presence of non-IID data.

The remainder of the chapter is structured as follows. Section 5.2 will report a background overview of the presented association rules-based detector. Section 5.3 will describe the

proposed federated architecture. Finally, Section 5.4 will discuss the experimental results.

5.2 BACKGROUND

Since a sequence of API calls can be effectively used to model the most representative behavioral features associated with a specific malware application, the background concepts related to the association rules-based detector, presented in [67], are recalled in this Section. More precisely, we first report its workflow through a step-by-step example. Then, we provide some mathematical definitions related to the rules pruning phase, which is essential to obtain relevant classification results.

5.2.1 *Association rules-based detector*

The execution flow of a specific application t_m can be represented as a sequence of API calls. As a consequence, specific rules able to represent the given application can be extracted in the form $\{API_i \rightarrow API_j\}$, with $API_i \prec API_j$. Note that such rules could include API calls not necessarily contiguous. The number of API calls skipped is considered to be the spacing of the rule. After that, such rules can be associated with nodes of a Markov chain to represent the application as a sequence of independent state transitions [9, 164]. Ultimately, we can describe such an API calls flow (i.e. a specific application) by using a graph, with nodes representing two (not necessarily contiguous) API calls and edges identifying their transition timeline (the sequence of invocation) [8]. In order to mine any possible insight from the API calls sequence, different pairs of API calls (rules) are extracted by varying the spacing.

Hence, with a little abuse of notation needed for simplification purposes, the related training process consists, for each application t_m, of n progressive steps, with $k \in [1, \ldots, n]$ representing the spacing in terms of positions to be skipped within the API calls sequence.

For each intermediate step k (with $k < n$), all the $k-1$ spaced transitions between two API calls are arranged in a Markov-like

chain represented by a graph $G_k^{t_m}$, in which the x-th node, N_x, is defined as follows:

$$N_x = [(API_i \rightarrow API_j), \sigma_x] \tag{5.1}$$

where $(API_i \rightarrow API_j)$ represents the mined rule and σ_x is the related number of occurrences, while the edges represent the transition states.

Next, in the last step ($k = n$), all the graphs are merged into one, namely G^{t_m}, representing the entire execution profile of the application t_m. Also, both the edges and the total number of occurrences of each node x of G^{t_m}, i.e. $\sigma^{t_m}(x)$, are updated. In particular, the occurrences are computed by summing the occurrences σ_x of the same nodes of the previous $(n-1)$ graphs.

For instance, let $t_1 = ADDDBCDD$ and $t_2 = ADDBCCCC$ be the API call sequences of two applications, by assuming only three steps (n = 3), Figure 5.1 and Figure 5.2 show the extracted graphs $G_k^{t_m}$ of each analyzed application at time-step $k = 1$ and $k = 2$, respectively. As depicted, at time-step $k = 1$, the transitions considered are contiguous, and no elements are present between two API calls used for mining a rule. Contrary, at the time-step $k = 2$, every rule is extracted by skipping an API call.

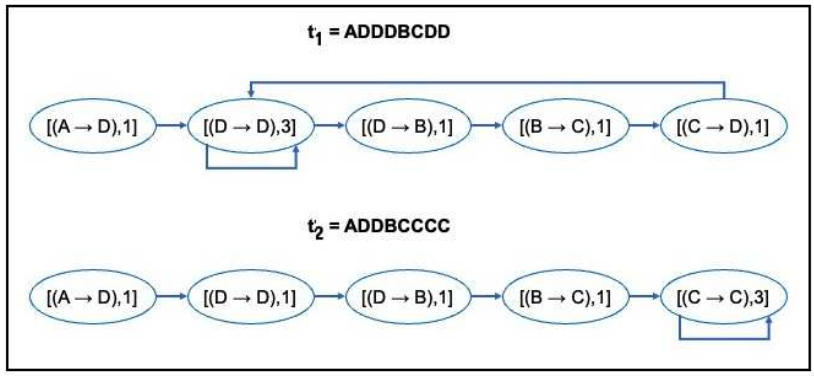

Figure 5.1: Extracted graphs $G_k^{t_1}$ and $G_k^{t_2}$ at time-step $k = 1$.

Next, at time-step $k = 3$, the previous graphs are merged into a new graph G^{t_m} representing the Run-time behavior of a considered application, as shown in Figure 5.3.

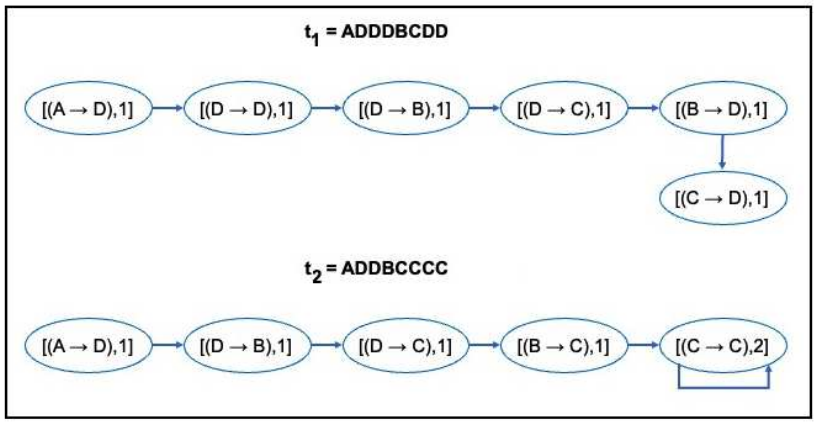

Figure 5.2: Extracted graphs $G_k^{t_1}$ and $G_k^{t_2}$ at time-step $k = 2$.

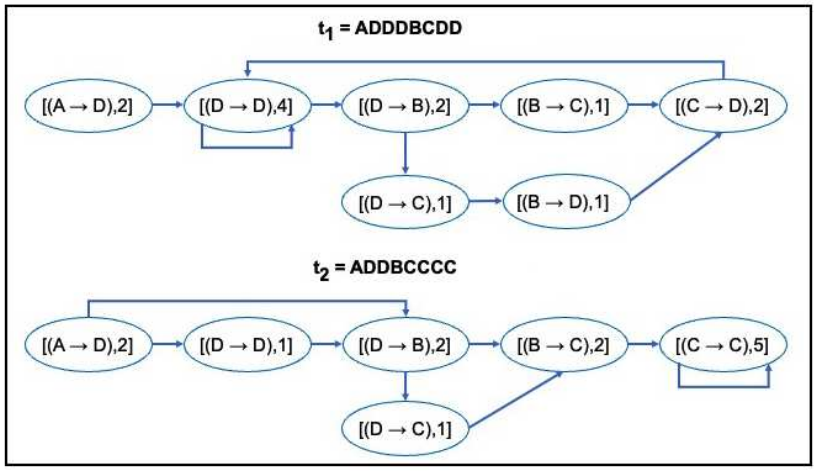

Figure 5.3: Merged graphs G^{t_1} and G^{t_2} related to t_1 and t_2 when $k = 3$.

Finally, since the set of association rules mined from different applications can be used as a signature characterizing a specific malware category (or class) $c \in C$ (where $C = \{c_1, \ldots, c_{|C|}\}$ is the set of malware categories), all the graphs G^{t_m} associated with $\mathcal{N}(c)$ applications belonging to the class c of the training dataset T, are further merged as a final graph G_c, in which the total number of occurrences of each node, σ_c, is again updated as well as the related edges, as shown in Figure 5.4. Note that, as better explained below (see Equation 5.3), σ_c is estimated by taking into account the different lengths (in terms of the number of

API calls) of the applications. Therefore, the proposed detector is characterized by a time complexity of $O(|T| \times n \times l)$, where $|T|$ is the training dataset dimension, n is the number of rules extraction-related steps, and l is the number of API calls.

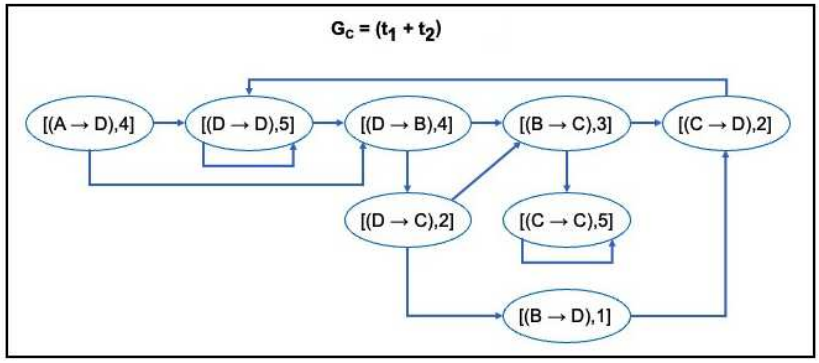

Figure 5.4: Final graph G_c derived by merging t_1 and t_2.

5.2.2 *Pruning phase definition*

Many rules could be extracted from the training process, thus a pruning step is of paramount importance to select only relevant information and then achieve relevant classification results. It is performed by comparing the occurrence of each rule with a threshold value. For instance, by considering a threshold of 2, all the rules with a $\sigma_x \leq 2$ could be pruned, as shown in red in Figure 5.5.

However, the existence of similar rules among different malware families could lead to incorrect classification results. To address this issue, a more complex pruning phase is considered, that is only rules whose values of support and confidence that satisfy a given property (e.g., associated with a specific threshold value) are considered valid.

More precisely, let $A = \{a_1, ..., a_{|A|}\}$ the set of admissible API calls, a generic rule is defined as follows:

$$R_{pq} = \{a_p \rightarrow a_q\} \tag{5.2}$$

with $a_p, a_q \in A$ and $a_p \prec a_q$, which denotes that the API a_p is called before a_q.

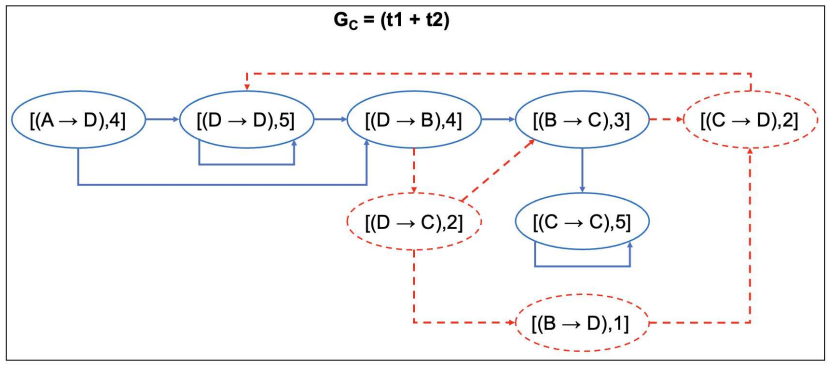

Figure 5.5: Pruning phase of rules with $\sigma_x \leq 2$.

By recalling the aforementioned association between rules and nodes in the merged graph G^{t_i} associated with the application $t_i \in T$, and defining $\sigma^{t_i}(R_{pq})$ as the number of occurrence of the rule R_{pq} in a given application $t_i \in T$, then the support of the rule R_{pq} with respect to the class c is defined as follows:

$$\Phi^c(R_{pq}) = \frac{|R_{pq}| \sum_{i=1}^{\mathcal{N}(c)} \frac{\sigma^{t_i}(R_{pq})}{l_i}}{\mathcal{N}(c)} \tag{5.3}$$

where l_i is the number of API calls of t_i, while $|R_{pq}|$ is the cardinality of the rule (i.e. 2).

Notice that, to take into account the different APIs flow lengths that could occur among the applications of a given class c as well as the unbalancing among the applications falling within classes of T, the terms l_i, $|R_{pq}|$, and $\mathcal{N}(c)$ of Equation 5.3 are used to normalize the support within the range $[0, 1]$.

However, as it is known, the support is not sufficient to estimate the quality of the rules in representing the applications for multi-class contexts. Thus, the confidence of a given rule R_{pq} on a class c is also defined, as follows:

$$\Gamma^c(R_{pq}) = \frac{\Phi^c(R_{pq})}{\sum_{v \in C} \Phi^v(R_{pq})} \tag{5.4}$$

Equation 5.4 expresses the ability of a rule to be unique for a specific class. Indeed, high values denote high uniqueness, while low values indicate that the rule is also present in other malware classes.

As depicted, Equation 5.3 and Equation 5.4 express a metric characterizing a given rule concerning a class c, and thus, they can be involved in the pruning phase. More specifically, rules having support and confidence less than given thresholds are pruned.

5.2.3 *Classification*

Once the training phase is performed and, thus, several rules are obtained, the classification of new incoming applications needs to be implemented. To accomplish this, the following metrics are introduced [67].

Firstly, the following confidence is provided:

$$\Gamma^{c}_{R_{pq}}(t_m) = \begin{cases} 0 & R_{pq} \not\subseteq t_m, \\ \sigma^{t_m}(R_{pq}) * \Gamma^{c}(R_{pq}) & R_{pq} \subseteq t_m \ \wedge \ \gamma(t_m) = c, \\ 1/\Gamma^{c}(R_{pq}) & R_{pq} \subseteq t_m \ \wedge \ \gamma(t_m) \neq c. \end{cases} \tag{5.5}$$

where $\gamma(t_m)$ is the hypothesis of membership classes associated with t_m.

It represents the confidence of a rule R_{pq} with respect to a class c associated with an application t_m. As shown, its value depends on the presence of the rule R_{pq} within the application t_m, as well as on the value assumed by the hypothesis of membership classes.

After that, the degree of belonging to class c of an application t_m is estimated by evaluating a rank ρ, which is given by:

$$\rho^{c}(t_m) = \sum_{c \in C} \sum_{\forall p,q} \Gamma^{c}_{R_{pq}}(t_m), \tag{5.6}$$

where the summation on p and q takes into account all the rules derived from the training phase.

Finally, the softmax function is used to classify the application t_m, as follows:

$$\gamma(t_m) = \arg\max_{h \in C} \frac{e^{\rho^{h}(t_m)}}{\sum_{v \in C} e^{\rho^{v}(t_m)}}. \tag{5.7}$$

5.3 THE PROPOSED ARCHITECTURE

This section presents the proposed federated architecture. More precisely, we extend the support and confidence indexes within a privacy-preserving federated environment. Then, we provide some implementation details related to core modules. Finally, we describe the proposed schema by highlighting its main advantages.

5.3.1 *Federated indexes definition*

The first step necessary for implementing the previously-discussed detector within a federated logic is to extend both support and confidence by considering the entire dataset T split among several federated clients. Note that different clients could deal with similar malware. As a consequence, T could include multiple copies of the same applications.

Let M be the number of considered clients, and T_j be the j-th dataset gathered by the client j, then the entire dataset T is subject to the following:

$$T = \bigcup_{j=1}^{M} T_j \tag{5.8}$$

We remark that, during the learning process, no T_j dataset is sent to the central server, but only the applications-related graphs G_c are sent, guaranteeing the protection of privacy. Furthermore, Equation 5.8 expresses the ability of our approach to tackle the problem of learning from non-IID data. Notice that this is true because according to Equation 5.8, all partial graphs are merged into a single one, and then the support and confidence are evaluated only by the central server.

Accordingly, let $\mathcal{N}(c, T_j)$ be the number of application $t_i \in T_j$ whose class is c, then the federated support of the rule R_{pq} can be defined as follows:

$$\Phi_f^c(R_{pq}) = \frac{|R_{pq}| \sum_{j=1}^{M} \sum_{i=1}^{\mathcal{N}(c,T_j)} \frac{\sigma^{t_i}(R_{pq})}{l_i}}{\mathcal{N}(c)} \tag{5.9}$$

Similarly, the federated confidence of a given rule R_{pq} on a class c is defined as follows:

$$\Gamma_f^c(R_{pq}) = \frac{\Phi_f^c(R_{pq})}{\sum_{v \in C} \Phi_f^v(R_{pq})} \tag{5.10}$$

Finally, Equation 5.5 and Equation 5.6 can be easily generalized to the federated case by replacing Φ with Φ_f and Γ with Γ_f, respectively and, also in this case, the classification of an application is performed by Equation 5.7.

5.3.2 *The federated rules-based detector*

To overcome the aforementioned confidentiality and privacy issues related to sharing data, we present an architecture that aims to support malware classification tasks by embedding the proposed associative rules-based detector within a federated logic, in which the involved IoT devices need to send their data to a central aggregation point devoted to sharing information. More precisely, the proposed architecture aims to provide a privacy-preserving data aggregation workflow where federated entities share only their applications-related graphs (and not the raw data) and receive the malware detector. Thus, the architecture is able to avoid the interchange of sensitive data among clients, and between clients and the server. Therefore, clients' privacy is fully guaranteed like in traditional Federated Learning-based approaches. Besides, our approach can avoid the well-known issues associated with the integration operations. Indeed, the centralized aggregation of graphs is extremely simple and presents a low computational effort. Note that, since Markov chains are defined as a memoryless stochastic process, each set of mined k-spaced associative rules, derived from the dynamic analysis of API calls, represents a locally trained model, which is equivalent to the local model of the classic FL-based solutions.

Moreover, the proposed workflow needs (in theory) only one centralized aggregation to build the presented classifier, and that does not need any usage of sophisticated algorithms, such as the Federated Averaging (FedAvg) [165], Federated Matched Averaging (FedMA) [166], and Federated Distance (FedDist) [167]. Therefore, the obtained results depend only on applications processed

by the federated entities. In addition, since the defined support and confidence indexes (see Equation 5.9 and Equation 5.10) are computed on the centralized and aggregated associative rules, we highlight the ability of the proposed logic to overcome the non-IID data-related issues that, instead, adversely affect the traditional federated learning-based solutions. For this reason, the presented workflow differs from pure FL-based ones and is also extremely suitable to improve the convergence process among Edge and Cloud infrastructures, with specific reference to data aggregation, data security, and services migration [168–170]. Hence, to report as much detailed information as possible, we describe the resulting architecture, structured according to a Publish-Subscribe model/policy, by defining three processes named *Client-Side Extraction*, *Server-Side Aggregation*, and *Detector Update*, respectively.

Therefore, the main goals of the proposed architecture can be summarized as follows:

1. A data extraction workflow is needed to collect associative rules from each federated entity (Client-Side Extraction);
2. A data aggregation workflow is useful to manage the received rules as category graphs and share a malware detector with each entity (Server-Side Aggregation);
3. A data update workflow is necessary for periodically re-adapting and re-sharing the malware detector (Detector Update).

5.3.2.1 *Client-Side Extraction process*

At the beginning of the Client-Side Extraction process, each federated entity asks to subscribe to the central server and receives the number of steps n to run. Next, for each analyzed application, and at each iteration $k < n$, the client extracts the k-spaced associative rules. Finally, the client sends the related graph to the central server. Figure 5.6 reports the discussed Client-Side Extraction process, while its steps can be summarized as Algorithm 1.

More precisely, for each application t, Algorithm 1 derives the corresponding class c and the list of API calls (*apisList*). Next, at

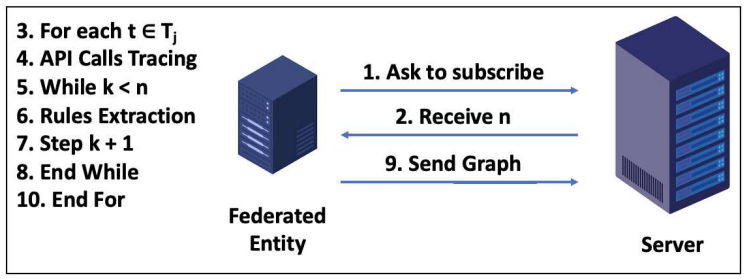

Figure 5.6: Client-Side Extraction process.

Algorithm 1 Client-Side Extraction

Require: T_j (j-th dataset of applications)
1: $n \leftarrow$ askSubscription()
2: **for each** t $\in T_j$ **do**
3: k $\leftarrow 1$
4: G $\leftarrow \emptyset$
5: c $\leftarrow$ Class(t)
6: apisList $\leftarrow$ traceAPIs(t)
7: **while** k $< n$ **do**
8: G $\leftarrow$ extractRules(apisList,k)
9: $k \leftarrow k + 1$
10: **end while**
11: sendGraph(G,c)
12: **end for**

each iteration $k < n$, the algorithm extracts the set of k-spaced associative rules by storing them in G. Note that, after the while loop, G will represent the graph containing any mined rules. Finally, the algorithm sends G and c to the central server and processes another application.

5.3.2.2 *Server-Side Aggregation process*

The Server-Side Aggregation process has the fundamental task of collecting the application-related graphs to share the proposed model with each federated entity. To accomplish this, the server first merges each graph with those previously received. Then, it uses the obtained information (i.e. the learned rules) to perform the pruning phase and share the malware detector. Note that the

described process confers to the server the capability of continuously sharing an updated classifier each time a new application is received. Therefore, the server can immediately support the federated entities for their detection activities, also in presence of zero-day malware. Figure 5.7 shows the Server-Side Aggregation process, while its steps can be summarized as Algorithm 2.

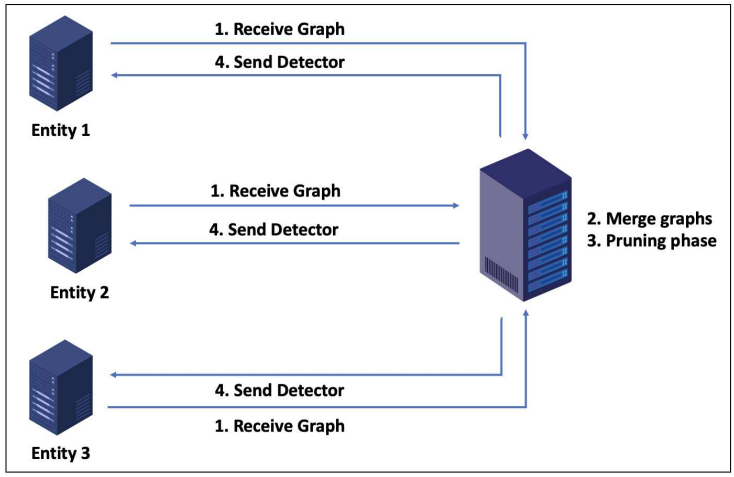

Figure 5.7: Server-Side Aggregation process.

Algorithm 2 Server-Side Aggregation

Require: M (number of subscribed clients)

1: **for each** i $\leq$ M **do**
2: (G,c) $\leftarrow$ receiveGraph(i)
3: graphList[c] $\leftarrow$ mergeGraph(G)
4: detector $\leftarrow$ pruningPhase(graphList)
5: sendToClients(detector)
6: **end for**

More precisely, every time the i-th client sends the graph G and the corresponding class c, Algorithm 2 merges G with the graphs previously stored in $graphList[c]$. Note that $graphList[c]$ contains the set of k-spaced rules related to class c. Next, the algorithm performs the pruning phase and sends the *detector* to each subscribed client.

5.3.2.3 *Detector Update process*

This process is responsible for taking into account new malware applications when a detector has already been shared. It combines the previous phases by considering new applications. We point out the ability of the presented architecture to continuously share an updated classifier regardless of the presence of non-IID data. When an unknown malware application is detected from a federated entity, the application-related graphs are built and sent to the server along with the related malware class. Next, the server will merge the received graphs with those already stored, and after the pruning phase, it will share the updated malware detector with each subscribed entity.

5.4 EXPERIMENTAL RESULTS

The first goal of the experiments is devoted to demonstrating the contribution of the proposed architecture concerning the classification of several malware applications, while the second is to study the required computational effort through performance analysis. To accomplish this, we first show the effectiveness of the proposed malware detector, also comparing it with other state-of-the-art ML-based approaches, by considering both centralized and decentralized data. Next, we analyze the required computational effort in presence of several federated entities and non-IID data partitions, respectively.

5.4.1 *Dataset and Experimental setting*

The dataset considered in the following experiments has been derived by Unisa Malware Dataset (UMD) [171], composed of about 3500 applications grouped in 8 Android families: Airpush (Air), DroidKungFu (DKF), Fusob (Fus), Genpua (Gen), GinMaster (Gin), Jisut (Jis), Opfake (Opf), and SmsPay (Sms). More precisely, to obtain the related API call sequences, we have analyzed each application through the Cuckoo Sandbox tool [172], which performs both static and dynamic malware analysis. Next, to evaluate the performances of the proposed detector, we have divided

the resulting dataset into training and testing sets according to the 70/30 criteria, as reported in Table 5.1.

Family	Training	Testing	Total
Airpush	265	114	379
DroidKungFu	700	301	1001
Fusob	117	49	166
Genpua	220	94	314
GinMaster	372	160	532
Jisut	376	161	537
Opfake	431	184	615
SmsPay	122	52	174
Total	2603	1115	3718

Table 5.1: Dataset division according to the 70/30 criteria.

Then, we used the obtained sets in both centralized and decentralized learning scenarios by respectively considering several data sub-partitions, splitting criteria, and the number of clients:

- Splitting: Horizontal (Samples of each category equally distributed), Vertical (Samples of a category are assigned to some clients only), and Mixed;
- Clients: 4, 8, 12, and 16.

Finally, we have simulated the proposed architecture within a socket-based Client-Server scenario, empowered by Python Remote Objects [173], by using a MacBook Pro equipped with an Apple M1 CPU and 16 GB of unified memory for the sever-side and several Linux-based virtualized clients equipped with 2 Processors and 2 GB RAM, respectively.

5.4.2 *Evaluation metrics*

To appreciate the classification quality of the proposed federated detector, we used the following evaluation metrics derived from the multi-class confusion matrix: Accuracy (Acc.), Sensitivity

(Sens.), Specificity (Spec.), Precision (Prec.), F-Measure (F-Mea.), and Area Under the ROC Curve (AUC).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.11}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{5.12}$$

$$Specificity = \frac{TN}{TN + FP} \tag{5.13}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.14}$$

$$F - Measure = \frac{2 * Sens * Prec}{Sens + Prec} \tag{5.15}$$

$$AUC = \frac{Sens + Spec}{2} \tag{5.16}$$

For each category, TPs (True Positives) are the applications correctly classified, while TNs (True Negatives) are the applications correctly identified in another category. Conversely, FPs (False Positives) are the applications incorrectly identified as a considered category, while FNs (False Negatives) are the malware applications in another category incorrectly identified as a considered category. In order to achieve a global perspective of the detector effectiveness, also the average performance values (Avg.) among all the observed malware classes have been computed.

5.4.3 *Achieved results*

To demonstrate the effectiveness and evaluate the performances of the proposed federated architecture, at first we considered the aforementioned dataset as centralized data by processing the training set with one client only. For each application, all the possible associative rules have been mined by using a progressive spacing $k \in [1, 100]$, and thus by sending them to the server as related graphs. Then, we merged, for each malware family, the stored graphs as one representative graph. Next, since the number of extracted rules is too high and could adversely affect the classification results, we have performed the pruning phase by considering any possible value of support (supp.) and confidence (conf.) between 0.1 and 1.0. Finally, we have used the

testing set to evaluate the quality of the selected training rules by achieving the best classification performances for the combination of supp./conf. = 0.7/1.0. The obtained results, shown in Table 5.2 and Table 5.3, demonstrate the abilities of the detector in classifying several malware families with an average accuracy of 99% and a limited number of FPs and FNs.

	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
Air	112	0	0	1	1	0	0	0
DKF	4	289	0	2	1	0	0	5
Fus	0	0	49	0	0	0	0	0
Gen	1	0	0	89	1	0	0	3
Gin	1	3	1	1	153	0	0	1
Jis	0	0	0	0	0	161	0	0
Opf	0	0	0	0	0	0	184	0
Sms	1	1	0	6	1	0	0	43

Table 5.2: Confusion matrix related to centralized data.

Family	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Air	0.9917	0.9412	0.9979	0.9825	0.9696	0.9614
DKF	0.9854	0.9863	0.9851	0.9601	0.9857	0.9731
Fus	0.9991	0.9800	1.0000	1.0000	0.9900	0.9899
Gen	0.9863	0.8990	0.9950	0.9468	0.9470	0.9223
Gin	0.9899	0.9745	0.9925	0.9563	0.9835	0.9653
Jis	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Opf	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Sms	0.9836	0.8269	0.9914	0.8269	0.9092	0.8269
Avg.	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549

Table 5.3: Performance results related to centralized data.

Subsequently, in order to show the learning abilities in a federated environment, we have repeated the training process by

considering several training set sub-partitions distributed, each time, among different clients. As shown in Table 5.4, since the application-related graphs extraction process is performed only by clients, the obtained results show how the proposed federated detector model can achieve excellent classification performance, while preserving privacy, independently of the splitting criteria used and the number of subscribed clients.

Clients	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Centr.	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
4	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
8	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
12	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
16	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549

Table 5.4: Comparison with a different number of clients and splitting criteria (only Avg. values are reported).

5.4.4 *Comparison and discussion*

Next, to highlight the potentialities of the presented classifier, we first compared the achieved results with those derived by common ML-based methods provided by Scikit-learn [174] (whose implementation is publicly available), namely: the Random Forest (RF) algorithm, Linear Support Vector Machine (SVM) classifier, Decision Trees (DTs), and Gaussian Naive Bayes (GNB) classifier, respectively. These methods have specifically been chosen because of their effectiveness in classification runtime performance. We accomplished this by arranging the k-spaced APIs sequences as corresponding adjacent matrices, in which each APIs pair has been represented with the related frequency. Table 5.5, Table 5.6, Table 5.7, and Table 5.8 report the obtained classification metrics for each ML-based method used, while Table 5.9 compares these results with those derived by our approach.

As reported in Table 5.9, the proposed approach outperformed each traditional ML-based method taken into consideration. More precisely, it achieved an average F-Measure improvement of 19%

Family	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Air	0.9748	0.9023	0.9844	0.8844	0.9434	0.8933
DKF	0.9474	0.9494	0.9468	0.8513	0.9481	0.8977
Fus	0.9998	1.0000	0.9998	0.9969	0.9999	0.9985
Gen	0.9677	0.8063	0.9871	0.8830	0.8967	0.8429
Gin	0.9681	0.8757	0.9869	0.9318	0.9313	0.9029
Jis	0.9988	0.9979	0.9990	0.9934	0.9984	0.9956
Opf	0.9970	0.9900	0.9978	0.9822	0.9939	0.9861
Sms	0.9690	0.6348	0.9902	0.8034	0.8125	0.7093
Avg.	0.9778	0.8946	0.9865	0.9158	0.9405	0.9033

Table 5.5: Performance results related to Random Forest.

Family	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Air	0.9682	0.8820	0.9797	0.8517	0.9308	0.8666
DKF	0.9422	0.9042	0.9544	0.8642	0.9293	0.8837
Fus	0.9973	1.0000	0.9971	0.9585	0.9986	0.9788
Gen	0.9557	0.7907	0.9756	0.7960	0.8832	0.7933
Gin	0.9556	0.8578	0.9755	0.8770	0.9167	0.8673
Jis	0.9978	0.9901	0.9989	0.9929	0.9945	0.9915
Opf	0.9931	0.9736	0.9955	0.9639	0.9846	0.9688
Sms	0.9525	0.4810	0.9824	0.6346	0.7317	0.5472
Avg.	0.9703	0.8599	0.9824	0.8674	0.9212	0.8622

Table 5.6: Performance results related to Linear SVM.

on the GNB classifier and 11% on the Decision Trees. In addition, the RF and SVM classifiers, which have also achieved good classification results, have been outperformed by our federated detector with an average improvement of 5% and 9%, respectively.

However, we also point out that, differently from our approach, all the other methods used in the comparison have been trained on the centralized raw data arranged as adjacent matrices, and thus they need an extra pre-processing step as well as do not

guarantee the privacy of the involved clients and confidentiality of related data.

In addition, the adjacent matrices (73×73) have been used to test also a DL-based approach. We accomplished this through a Keras API-based CNN, shown in Figure 5.8, trained in both centralized and federated-learning scenarios. Table 5.10 summarizes the results derived by considering each splitting criteria and the number of subscribed clients, Table 5.11 reports the client-related Accuracy values derived on the testing dataset, while Table 5.12 compares our approach with the above CNN.

Family	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Air	0.9516	0.8116	0.9702	0.7828	0.8909	0.7969
DKF	0.9336	0.8715	0.9535	0.8574	0.9125	0.8644
Fus	0.9986	1.0000	0.9985	0.9786	0.9993	0.9892
Gen	0.9461	0.7297	0.9721	0.7591	0.8509	0.7441
Gin	0.9489	0.8473	0.9696	0.8501	0.9084	0.8487
Jis	0.9982	0.9979	0.9983	0.9888	0.9981	0.9933
Opf	0.9936	0.9843	0.9948	0.9586	0.9895	0.9713
Sms	0.9519	0.5201	0.9792	0.6135	0.7496	0.5629
Avg.	0.9653	0.8453	0.9795	0.8486	0.9124	0.8464

Table 5.7: Performance results related to Decision Trees.

As shown in Table 5.10, only with the Mixed splitting (Mix), the CNN achieved comparable results with those derived from the centralized training dataset. Their effectiveness is also confirmed by Table 5.11, which reports similar local-related Accuracy values for each number of subscribed clients. Vice versa, due to the lack of convergence of the Federated CNN, no results have been achieved by the remaining two splitting criteria (Vertical and Horizontal).

More precisely, the Vertical one has highlighted how DL-based models cannot learn if the data-related categories are distributed only to some clients, adversely affecting the federated learning process. Instead, the Horizontal one has highlighted how IID data can also adversely influence the FL-based models. This occurs when data are characterized from several sub-categories because

Family	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Air	0.9432	0.7546	0.9682	0.7587	0.8614	0.7567
DKF	0.8760	0.7551	0.9149	0.7401	0.8350	0.7475
Fus	0.9925	1.0000	0.9920	0.8935	0.9960	0.9438
Gen	0.9449	0.6280	0.9830	0.8165	0.8055	0.7100
Gin	0.9033	0.5411	0.9771	0.8280	0.7591	0.6545
Jis	0.9850	0.9994	0.9828	0.8984	0.9911	0.9462
Opf	0.9359	0.9811	0.9304	0.6334	0.9558	0.7698
Sms	0.9541	0.5894	0.9772	0.6215	0.7833	0.6050
Avg.	0.9419	0.7811	0.9657	0.7738	0.8734	0.7667

Table 5.8: Performance results related to Gaussian NB.

Method	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Proposed	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
RF	0.9778	0.8946	0.9865	0.9158	0.9405	0.9033
SVM	0.9703	0.8599	0.9824	0.8674	0.9212	0.8622
DTs	0.9653	0.8453	0.9795	0.8486	0.9124	0.8464
GNB	0.9419	0.7811	0.9657	0.7738	0.8734	0.7667

Table 5.9: Comparison with ML-based methods (only Avg. values are reported).

Split	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Centr.	0.9697	0.8692	0.9820	0.8741	0.9256	0.8711
Mix4	0.9695	0.8705	0.9819	0.8754	0.9262	0.8714
Mix8	0.9659	0.8585	0.9799	0.8576	0.9192	0.8563
Mix12	0.9635	0.8467	0.9784	0.8486	0.9126	0.8462
Mix16	0.9603	0.8318	0.9766	0.8325	0.9042	0.8297

Table 5.10: Performance Results related to CNN (only Avg. values are reported).

they produce a sort of nested Vertical splitting. In our case, this means that each k-spaced adjacent matrix can be related

Clients	Split	min_Acc.	max_Acc.	avg_Acc.
4	Mix	0.9678	0.9687	0.9683
8	Mix	0.9638	0.9655	0.9647
12	Mix	0.9595	0.9622	0.9613
16	Mix	0.9561	0.9595	0.9583

Table 5.11: Clients-related Accuracy values.

Method	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Proposed	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
Centr.	0.9697	0.8692	0.9820	0.8741	0.9256	0.8711
Mix4	0.9695	0.8705	0.9819	0.8754	0.9262	0.8714

Table 5.12: Comparison between our approach and the CNN (only best values are reported).

to a specific k-th sub-category, with k varying from 1 to 100. Therefore, these matrices adversely affect the federated learning process also if data are Independent and Identically Distributed (IID). Finally, the comparison provided in Table 5.12 shows that our approach outperformed the FL-based CNN by achieving an average F-Measure improvement of 8%. Therefore, this confirms the effectiveness of the proposed detector in achieving excellent classification metrics for any considered splitting criteria (IID and non-IID) that, instead, adversely affect the classical FL-based solutions [20, 82–84].

5.4.5 *Performance evaluation*

To evaluate the performance of the proposed architecture during the Client-Side Extraction and Server-Side Aggregation processes, we first derived the required time effort by using the entire training set on a single machine. Next, we partitioned it by applying the previously mentioned splitting criteria and considering the number of involved clients, as reported in Table 5.13, Table 5.14, Table 5.15, Table 5.16, Table 5.17, Table 5.18, Table 5.19, Table 5.20 and Table 5.21, respectively.

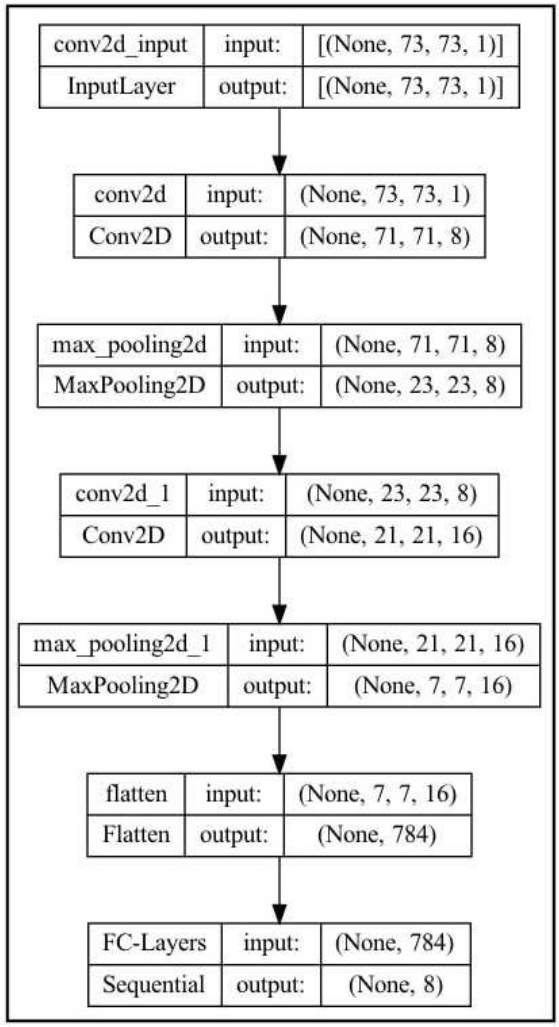

Figure 5.8: Architecture of the CNN used in comparisons.

Hence, to carefully analyze the model scalability for each considered combination (splitting criteria/number of clients), we derived the parallel Speedup, which is given by:

$$Speedup = \frac{T_s}{T_p}, \tag{5.17}$$

where T_s is the time effort without parallelism and T_p is the required time effort with parallelism. Figure 5.9 shows the speedup-

related behavior for each considered splitting criteria and clients, respectively.

As shown in Figure 5.9, since the derived Speedup values are slightly different, it is possible to appreciate how the proposed model is characterized by semi-linear scalability. More precisely, when the number of clients is low (i.e., 4 and 8), the system rapidly scales thanks to low communication costs. Instead, when the number of clients increases, the related execution costs decrease to the point that counterbalances the high communication costs. For this reason, differently, by the classical FL-based solutions [20, 82–84], the proposed architecture can obtain excellent time performances in the presence of non-IID.

Clients	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
4	66	175	29	55	93	94	107	30
8	33	87	14	27	46	47	53	15
12	22	58	9	18	31	31	35	10
16	16	43	7	13	23	23	26	7

Table 5.13: Horizontal training set division for each client.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	265	700	0	0	0	0	0	0
C2	0	0	117	220	0	0	0	0
C3	0	0	0	0	372	376	0	0
C4	0	0	0	0	0	0	431	122

Table 5.14: Vertical training set division for 4 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	265	350	0	0	0	0	107	0
C2	0	350	117	0	0	0	107	0
C3	0	0	0	110	186	188	107	0
C4	0	0	0	110	186	188	110	122

Table 5.15: Mixed training set division for 4 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	132	350	0	0	0	0	0	0
C2	0	0	58	110	0	0	0	0
C3	0	0	0	0	186	188	0	0
C4	0	0	0	0	0	0	215	61
C5	133	350	0	0	0	0	0	0
C6	0	0	59	110	0	0	0	0
C7	0	0	0	0	186	188	0	0
C8	0	0	0	0	0	0	216	61

Table 5.16: Vertical training set division for 8 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	132	175	0	0	0	0	107	0
C2	0	175	117	0	0	0	107	0
C3	0	0	0	55	93	94	107	0
C4	0	0	0	55	93	94	110	122
C5	133	175	0	0	0	0	0	0
C6	0	175	0	0	0	0	0	0
C7	0	0	0	55	93	94	0	0
C8	0	0	0	55	93	94	0	0

Table 5.17: Mixed training set division for 8 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	66	175	0	0	0	0	0	0
C2	0	0	29	55	0	0	0	0
C3	0	0	0	0	93	94	0	0
C4	0	0	0	0	0	0	215	61
C5	133	350	0	0	0	0	0	0
C6	0	0	59	110	0	0	0	0
C7	0	0	0	0	186	188	0	0
C8	0	0	0	0	0	0	108	30
C9	66	175	0	0	0	0	0	0
C10	0	0	29	55	0	0	0	0
C11	0	0	0	0	93	94	0	0
C12	0	0	0	0	0	0	108	31

Table 5.18: Vertical training set division for 12 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	66	175	0	0	0	0	53	0
C2	0	175	29	0	0	0	53	0
C3	0	0	0	55	93	94	53	0
C4	0	0	0	55	93	94	53	30
C5	133	175	0	0	0	0	0	0
C6	0	175	0	0	0	0	0	0
C7	0	0	0	55	93	94	0	0
C8	0	0	0	55	93	94	0	0
C9	66	0	0	0	0	0	53	0
C10	0	0	29	0	0	0	53	30
C11	0	0	29	0	0	0	53	30
C12	0	0	30	0	0	0	60	32

Table 5.19: Mixed training set division for 12 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	66	175	0	0	0	0	0	0
C2	0	0	29	55	0	0	0	0
C3	0	0	0	0	93	94	0	0
C4	0	0	0	0	0	0	107	30
C5	66	175	0	0	0	0	0	0
C6	0	0	29	55	0	0	0	0
C7	0	0	0	0	93	94	0	0
C8	0	0	0	0	0	0	107	30
C9	66	175	0	0	0	0	0	0
C10	0	0	29	55	0	0	0	0
C11	0	0	0	0	93	94	0	0
C12	0	0	0	0	0	0	107	30
C13	67	175	0	0	0	0	0	0
C14	0	0	30	55	0	0	0	0
C15	0	0	0	0	93	94	0	0
C16	0	0	0	0	0	0	110	32

Table 5.20: Vertical training set division for 16 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	66	87	0	0	0	0	53	0
C2	0	87	29	0	0	0	53	0
C3	0	0	0	55	93	94	53	0
C4	0	0	0	55	93	94	53	30
C5	66	87	0	0	0	0	0	0
C6	0	87	0	0	0	0	0	0
C7	0	0	0	55	93	94	0	0
C8	0	0	0	55	93	94	0	0
C9	66	0	0	0	0	0	53	0
C10	0	0	29	0	0	0	53	30
C11	0	0	29	0	0	0	53	30
C12	0	0	30	0	0	0	60	32
C13	67	87	0	0	0	0	0	0
C14	0	87	0	0	0	0	0	0
C15	0	87	0	0	0	0	0	0
C16	0	91	0	0	0	0	0	0

Table 5.21: Mixed training set division for 16 clients.

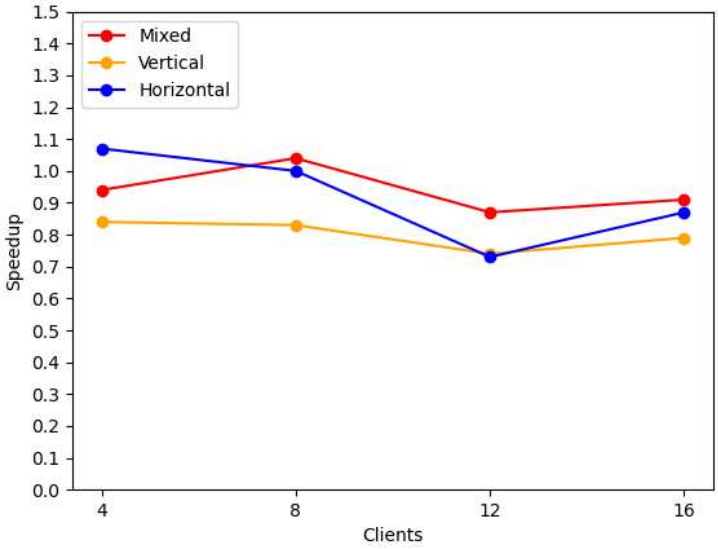

Figure 5.9: Speedup comparison.

6

CONCLUSION

In the field of network security, the increasing volume and complexity of data have made it challenging for traditional security systems to keep up with the changing security landscape. This is where the application of ML and DL methods has come in handy, offering the ability to handle vast amounts of data and detect security threats in real-time.

The use of ML algorithms has shown significant progress in detecting network intrusions, identifying malicious traffic, classifying network traffic, and improving the accuracy of security systems. For example, the application of unsupervised learning algorithms, like clustering, has been successful in detecting novel and unknown attacks that were previously undetectable. On the other hand, supervised learning algorithms like decision trees and random forests have proven to be effective in classifying network traffic and identifying specific security threats.

DL methods have also shown promising results in network security. CNNs and AEs have been used to analyze network traffic and identify security threats. RNNs have been used to analyze network logs and identify anomalies that may indicate security threats.

However, it is essential to note that these methods are not without their limitations. For instance, the accuracy of these algorithms is still not perfect and can lead to false positives and false negatives. Additionally, the ethical concerns related to privacy and data protection must be considered when using these methods.

In this thesis, we focused on the effectiveness of some dynamic analysis, ML, and DL-based strategies in dealing with crucial issues in network security and the related most disparate classification tasks (e.g., network traffic classification, encrypted traffic, anomaly and attack detection/classification, and malware detection).

In Chapter 4, we proposed a novel IoT-related network attacks classifier by combining the capabilities of AEs in automatically finding relevant features and the effectiveness of characteristics coming from non-linear analysis theory arranged as RPs. More precisely, we considered statistical information related to benign and malicious traffic-related activities available in the CIC-IDS2017 dataset as raw input data. Then, we employed the corresponding multi-channel image representation of RPs and a Convolutional Sparse Autoencoder (CNN-SAE) to perform automatic interpretation of RP images, aimed at using the reliable discriminating power of such non-linear structures in attack classification tasks. The achieved results, derived as statistic metrics from the multi-class confusion matrix, have proven the effectiveness of the proposed classifier in the presence of several unbalanced datasets partitions by obtaining an average F-Measure of more than 98%. Finally, we have evaluated the proposed approach by comparing it with the most famous state-of-the-art ML-based attack classification techniques such as Naive Bayes-based, Logistic, Support Vector Machine, J48 decision-tree, Multilayer Perceptron, and Convolutional Neural Network-based classifiers. The proposed approach outperformed all considered approaches with significant improvements in all evaluation metrics.

Therefore, based on the derived outcomes, we would like to propose two possible future works. The first one can be investigating the effectiveness of Recurrence Plots and CNN-SAEs for multiple hostile traffic flow detection and classification in federated organizations. The second one is exploring the abilities of Autoencoders in deriving relevant traffic features by considering the combination of several non-linear extraction methods. More precisely, the following studies might improve the effectiveness of real-time attack detectors and classifiers by reducing the impact of Distributed Denial of Services (DDoS) attacks and the required time for their detection, classification, and mitigation, respectively. By the way, these possible approaches can also be investigated by performing some comparisons with the advanced and existing methods in the state of the art in terms of effectiveness, robustness, and precision of evaluation metrics.

In Chapter 5, the capabilities of Markov chains and associations-based rules have been investigated within a federated environment in order to support privacy-aware malware detection and classification in an Android-based IoT environment. To accomplish this, we enhanced the dynamic-based detector, presented in [67], through a federated logic. Therefore, we proposed a dedicated architecture capable of performing a federated training process in which end-users build a shared detector by sending the analyzed applications to a central server. Consequently, the interchange of sensitive information, and their protection against the most common data leakage threats, has been similarly guaranteed as done in traditional Federated Learning-based approaches. Next, we validated the effectiveness of the presented method by employing several famous malware families derived from the UMD dataset. More precisely, the obtained results have shown an average accuracy of 99% by outperforming the most famous DL and FL-based approaches and highlighting possible benefits in zero-day malware detection. Finally, we also provided a statistical and temporal performance assessment in the presence of non-IID data, in which several dataset partitions, splitting criteria, and the number of subscribed clients have been considered, respectively.

However, due to the high number of existing and yearly released malware applications, this study proposes some possible future works that might provide some benefits, such as well-suited mechanisms to improve the zero-day detection and the building of new communication channels and environments for federated entities, respectively. For this reason, one can investigate an extension of the proposed detector in order to detect new malware families and variants characterized by more intricate dynamic patterns. For instance, given an arbitrary rule, the employed pruning indexes might consider only the "nearest rules". Also, as another direction, researchers can optimize the applications-related extraction process in order to improve the required temporal effort. For instance, several selection criteria could be employed (e.g., the proposed pruning indexes) to consider only the relevant rule sequences effectively mined during the application execution.

In general, and in conclusion, the application of ML, DL, and dynamic analysis-based methods in network security has shown

promising results in detecting, classifying, and mitigating security threats. While these methods still have room for improvement, they offer a valuable addition to the existing security landscape and provide a comprehensive approach to network security. Further research is necessary to improve these methods and find new ways to integrate them into existing security systems.

PUBLICATIONS

The contributions of this thesis can be found in the following publications:

[1] Gianni D'Angelo, Eslam Farsimadan, and Francesco Palmieri. "Recurrence Plots-based Network Attack Classification using CNN-Autoencoders." In: *Accepted in the 2023 International Conference on Computational Science and Its Applications (ICCSA 2023)* (2023).

[2] Gianni D'Angelo, Eslam Farsimadan, Massimo Ficco, Francesco Palmieri, and Antonio Robustelli. "Privacy-preserving malware detection in Android-based IoT devices through federated Markov chains." In: *Future Generation Computer Systems* (). ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2023.05.021.

[3] Gianni D'Angelo, Eslam Farsimadan, and Francesco Palmieri. "Recurrence Plots-based Network Traffic Classification using Non-linear Features and Convolutional Neural Networks." In: *Preparation* (2023).

BIBLIOGRAPHY

[1] ITU global ICT statistics. *Individuals using the Internet*. URL: https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx. (accessed: 25.11.2022).

[2] Douglas Jacobson. *Introduction to Network Security*. CRC Press, Taylor & Francis Group, 2008. ISBN: 978-1-4200-8587-7. URL: https://www.taylorfrancis.com/books/mono/10.1201/9781420010695/introduction-network-security-douglas-jacobson.

[3] Herjavecgroup. *herjavecgroup, Official annual cybercrime report, 2019*. URL: https://www.herjavecgroup.com/resources/2019-official-annual-cybercrime-report/. (accessed: 25.01.2020).

[4] CyberEdge Group. *Cyberthreat Defense Report (CDR) 2021*. URL: https://delinea.com/blog/key-takeaways-2021-cyberthreat-defense-report. (accessed: 01.02.2022).

[5] Tony Thomas, Athira P. Vijayaraghavan, and Sabu Emmanuel. *Machine Learning Approaches in Cyber Security Analytics*. Springer Nature Singapore Pte Ltd., 2020. ISBN: 978-981-15-1705-1. URL: https://doi.org/10.1007/978-981-15-1706-8.

[6] Valerian Rey, Pedro Miguel Sánchez Sánchez, Alberto Huertas Celdrán, and Gérôme Bovet. "Federated learning for malware detection in IoT devices." In: *Computer Networks* 204 (2022), pp. 1–14. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.108693.

[7] Francesco Palmieri and Ugo Fiore. "A nonlinear, recurrence-based approach to traffic classification." In: *Computer Networks* 53.6 (2009), pp. 761–773. DOI: https://doi.org/10.1016/j.comnet.2008.12.015.

[8] Massimo Ficco. "Detecting IoT Malware by Markov Chain Behavioral Models." In: *2019 IEEE International Conference*

on Cloud Engineering (IC2E). 2019, pp. 229–234. DOI: 10.1109/IC2E.2019.00037.

[9] Alejandro Martín, Víctor Rodríguez-Fernández, and David Camacho. "CANDYMAN: Classifying Android malware families by modelling dynamic traces with Markov chains." In: *Engineering Applications of Artificial Intelligence* 74 (2018), pp. 121–133. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2018.06.006.

[10] Gianni D'Angelo, Eslam Farsimadan, and Francesco Palmieri. "Recurrence Plots-based Network Attack Classification using CNN-Autoencoders." In: *Accepted in the 2023 International Conference on Computational Science and Its Applications (ICCSA 2023)* (2023).

[11] Gianni D'Angelo, Eslam Farsimadan, Massimo Ficco, Francesco Palmieri, and Antonio Robustelli. "Privacy-preserving malware detection in Android-based IoT devices through federated Markov chains." In: *Future Generation Computer Systems* (). ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2023.05.021.

[12] Gianni D'Angelo, Eslam Farsimadan, and Francesco Palmieri. "Recurrence Plots-based Network Traffic Classification using Non-linear Features and Convolutional Neural Networks." In: *Preparation* (2023).

[13] Sheraz Naseer, Yasir Saleem, Shehzad Khalid, Muhammad Khawar Bashir, Jihun Han, Muhammad Munwar Iqbal, and Kijun Han. "Enhanced Network Anomaly Detection Based on Deep Neural Networks." In: *IEEE Access* 6 (2018), pp. 48231–48246. DOI: 10.1109/ACCESS.2018.2863036.

[14] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. "Deep Learning for Anomaly Detection: A Review." In: *ACM Comput. Surv.* 54.2 (2021), pp. 1–38. ISSN: 0360-0300. DOI: 10.1145/3439950. URL: https://doi.org/10.1145/3439950.

[15] Jeetendra Pande. *Introduction to Cyber Security*. Uttarakhand Open University, 2017. ISBN: 978-93-84813-96-3. URL:

https://www.uou.ac.in/sites/default/files/slm/Introduction-cyber-security.pdf.

[16] Tibra Alsmadi and Nour Alqudah. "A Survey on malware detection techniques." In: *2021 International Conference on Information Technology (ICIT)*. 2021, pp. 371–376. DOI: 10.1109/ICIT52682.2021.9491765.

[17] Faitouri A Aboaoja, Anazida Zainal, Fuad A Ghaleb, Bander Ali Saleh Al-rimy, Taiseer Abdalla Elfadil Eisa, and Asma Abbas Hassan Elnour. "Malware detection issues, challenges, and future directions: A survey." In: *Applied Sciences* 12.17 (2022), pp. 1–29. DOI: https://doi.org/10.3390/app12178482.

[18] Zhiqiang Wang, Qian Liu, and Yaping Chi. "Review of Android Malware Detection Based on Deep Learning." In: *IEEE Access* 8 (2020), pp. 181102–181126. DOI: 10.1109/ACCESS.2020.3028370.

[19] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. *Federated Optimization in Heterogeneous Networks*. 2018. DOI: 10.48550/ARXIV.1812.06127. URL: https://arxiv.org/abs/1812.06127.

[20] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning." In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 5132–5143. DOI: https://doi.org/10.48550/arXiv.1910.06378.

[21] Dimitrios Papamartzivanos, Félix Gómez Mármol, and Georgios Kambourakis. "Introducing Deep Learning Self-Adaptive Misuse Network Intrusion Detection Systems." In: *IEEE Access* 7 (2019), pp. 13546–13560. DOI: 10.1109/ACCESS.2019.2893871.

[22] 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. 2009, pp. 1–6. DOI: 10.1109/CISDA.2009.5356528.

[23] Peng Jiang, Hongyi Wu, Cong Wang, and Chunsheng Xin. "Virtual MAC Spoofing Detection through Deep Learning." In: *2018 IEEE International Conference on Communications (ICC)*. 2018, pp. 1–6. DOI: 10.1109/ICC.2018.8422830.

[24] Sahil Garg, Kuljeet Kaur, Neeraj Kumar, Georges Kaddoum, Albert Y. Zomaya, and Rajiv Ranjan. "A Hybrid Deep Learning-Based Model for Anomaly Detection in Cloud Datacenter Networks." In: *IEEE Transactions on Network and Service Management* 16.3 (2019), pp. 924–935. DOI: 10.1109/TNSM.2019.2927886.

[25] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Uday Tupakula. "Autoencoder-based feature learning for cyber security applications." In: vol. 2017-May. Cited by: 199. 2017, pp. 3854 –3861. DOI: 10.1109/IJCNN.2017.7966342.

[26] Ritesh K. Malaiya, Donghwoon Kwon, Jinoh Kim, Sang C. Suh, Hyunjoo Kim, and Ikkyun Kim. "An Empirical Evaluation of Deep Learning for Network Anomaly Detection." In: *2018 International Conference on Computing, Networking and Communications (ICNC)*. 2018, pp. 893–898. DOI: 10.1109/ICCNC.2018.8390278.

[27] V. Thing. In: *Ieee 802.11 Network Anomaly Detection and Attack Classification: A Deep Learning Approach* (2017). Cited by: 1, pp. 1 –6. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85123413356&partnerID=40&md5=087c2ea0f41713f42ba4e71a3bc338ac.

[28] Yuanfang Chen, Yan Zhang, Sabita Maharjan, Muhammad Alam, and Ting Wu. "Deep Learning for Secure Mobile Edge Computing in Cyber-Physical Transportation Systems." In: *IEEE Network* 33.4 (2019), pp. 36–41. DOI: 10.1109/MNET.2019.1800458.

[29] Mohamed Abdel-Basset, Hossam Hawash, Ripon K. Chakrabortty, and Michael J. Ryan. "Semi-Supervised Spatiotemporal Deep Learning for Intrusions Detection in IoT Networks."

In: *IEEE Internet of Things Journal* 8.15 (2021), pp. 12251–12265. DOI: 10.1109/JIOT.2021.3060878.

[30] Nagarathna Ravi and S. Mercy Shalinie. "Semisupervised-Learning-Based Security to Detect and Mitigate Intrusions in IoT Network." In: *IEEE Internet of Things Journal* 7.11 (2020), pp. 11041–11052. DOI: 10.1109/JIOT.2020.2993410.

[31] Laisen Nie, Wentao Sun, Shupeng Wang, Zhaolong Ning, Joel J. P. C. Rodrigues, Yixuan Wu, and Shengtao Li. "Intrusion Detection in Green Internet of Things: A Deep Deterministic Policy Gradient-Based Algorithm." In: *IEEE Transactions on Green Communications and Networking* 5.2 (2021), pp. 778–788. DOI: 10.1109/TGCN.2021.3073714.

[32] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[33] 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)*. 2019, pp. 1–8. DOI: 10.1109/CCST.2019.8888419.

[34] Laisen Nie, Zhaolong Ning, Mohammad S. Obaidat, Balqies Sadoun, Huizhi Wang, Shengtao Li, Lei Guo, and Guoyin Wang. "A Reinforcement Learning-Based Network Traffic Prediction Mechanism in Intelligent Internet of Things." In: *IEEE Transactions on Industrial Informatics* 17.3 (2021), pp. 2169–2180. DOI: 10.1109/TII.2020.3004232.

[35] Xiaoding Wang, Sahil Garg, Hui Lin, Jia Hu, Georges Kaddoum, Md. Jalil Piran, and M. Shamim Hossain. "Toward Accurate Anomaly Detection in Industrial Internet of Things Using Hierarchical Federated Learning." In: *IEEE Internet of Things Journal* 9.10 (2022), pp. 7110–7119. DOI: 10.1109/JIOT.2021.3074382.

[36] Norbert Marwan, M. Carmen Romano, Marco Thiel, and Jürgen Kurths. "Recurrence plots for the analysis of complex systems." In: *Physics Reports* 438.5 (2007), pp. 237–329. ISSN: 0370-1573. DOI: https://doi.org/10.1016/j.physrep.2006.11.001. URL: https://www.sciencedirect.com/science/article/pii/S0370157306004066.

[37] Enrique Garcia-Ceja, Md. Zia Uddin, and Jim Torresen. "Classification of Recurrence Plots'Distance Matrices with a Convolutional Neural Network for Activity Recognition." In: *Procedia Computer Science* 130 (2018), pp. 157–163. DOI: https://doi.org/10.1016/j.procs.2018.04.025. URL: https://www.sciencedirect.com/science/article/pii/S1877050918303752.

[38] Nathalia Menini, Alexandre E. Almeida, Rubens Lamparelli, Guerric le Maire, Jefersson A. dos Santos, Helio Pedrini, Marina Hirota, and Ricardo da S. Torres. "A Soft Computing Framework for Image Classification Based on Recurrence Plots." In: *IEEE Geoscience and Remote Sensing Letters* 16.2 (2019), pp. 320–324. DOI: 10.1109/LGRS.2018.2872132.

[39] Manish S. Patil, Renuka Loka, and Alivelu M. Parimi. "Application of ARIMA and 2D-CNNs Using Recurrence Plots for Medium-Term Load Forecasting." In: *2021 IEEE 2nd China International Youth Conference on Electrical Engineering (CIYCEE)*. 2021, pp. 1–5. DOI: 10.1109/CIYCEE53554.2021.9676838.

[40] Lyudmyla Kirichenko, Tamara Radivilova, Vitalii Bulakh, Petro Zinchenko, and Abed Saif Alghawli. "Two Approaches to Machine Learning Classification of Time Series Based on Recurrence Plots." In: *2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)*. 2020, pp. 84–89. DOI: 10.1109/DSMP47368.2020.9204021.

[41] Ralph G. Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E. Elger. "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state." In: *Phys. Rev. E*

64 (6 2001), p. 061907. DOI: 10.1103/PhysRevE.64.061907. URL: https://link.aps.org/doi/10.1103/PhysRevE.64.061907.

[42] Konstantinos Tziridis, Theofanis Kalampokas, and George A. Papakostas. "EEG Signal Analysis for Seizure Detection Using Recurrence Plots and Tchebichef Moments." In: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. 2021, pp. 0184–0190. DOI: 10.1109/CCWC51732.2021.9376134.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[45] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. *Densely Connected Convolutional Networks*. 2016. DOI: 10.48550/ARXIV.1608.06993. URL: https://arxiv.org/abs/1608.06993.

[46] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

[47] Sara Sartoli, Yong Wei, and Shane Hampton. "Malware Classification using Recurrence Plots and Deep Neural Network." In: *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2020, pp. 901–906. DOI: 10.1109/ICMLA51294.2020.00147.

[48] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. *Microsoft Malware Classification Challenge*. 2018. DOI: 10.48550/ARXIV.1802.10135. URL: https://arxiv.org/abs/1802.10135.

[49] J.-P Eckmann, S. Oliffson Kamphorst, and D Ruelle. "Recurrence Plots of Dynamical Systems." In: *Europhysics Letters (EPL)* 4.9 (1987), pp. 973–977. DOI: 10.1209/0295-5075/4/9/004. URL: https://doi.org/10.1209/0295-5075/4/9/004.

[50] Yoshito Hirata. "Recurrence plots for characterizing random dynamical systems." In: *Communications in Nonlinear Science and Numerical Simulation* 94 (2021), p. 105552. ISSN: 1007-5704. DOI: https://doi.org/10.1016/j.cnsns.2020.105552. URL: https://www.sciencedirect.com/science/article/pii/S1007570420303828.

[51] Francesco Palmieri and Ugo Fiore. "Network anomaly detection through nonlinear analysis." In: *Comp. & Security* 29.7 (2010), pp. 737–755.

[52] Min Hu, Xiaowei Feng, Shengchen Zhou, and Wei Xu. "Anomaly Detection in Time Series Data Based on Unthresholded Recurrence Plots." In: *International Conference on Applications and Techniques in Cyber Security and Intelligence*. Springer. 2018, pp. 477–484.

[53] Pan Wang, Feng Ye, Xuejiao Chen, and Yi Qian. "Datanet: Deep Learning Based Encrypted Network Traffic Classification in SDN Home Gateway." In: *IEEE Access* 6 (2018), pp. 55380–55391. DOI: 10.1109/ACCESS.2018.2872430.

[54] Gianni D'Angelo and Francesco Palmieri. "Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction." In: *Journal of Network and Computer Applications* 173 (2021), p. 102890. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102890.

[55] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. "Network Traffic Classifier with Convolutional and Recurrent Neural Networks for Internet of Things." In: *IEEE Access* 5 (2017), pp. 18042–18050.

[56] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. "Deep packet: a novel approach for encrypted traffic clas-

sification using deep learning." In: *Soft Computing* 24.3 (2020), pp. 1999–2012. DOI: https://doi.org/10.48550/arXiv.1709.02656.

[57] Chet Hosmer. *Polymorphic and metamorphic malware - black hat briefings*. URL: https://www.blackhat.com/presentations/bh-usa-08/Hosmer/BH_US_08_Hosmer_Polymorphic_Malware.pdf.

[58] Kimberly Tam, Aristide Fattori, Salahuddin Khan, and Lorenzo Cavallaro. "CopperDroid: Automatic Reconstruction of Android Malware Behaviors." English. In: *NDSS Symposium 2015*. NDSS Symposium 2015. Feb. 2015, pp. 1–15. DOI: 10.14722/ndss.2015.23145.

[59] Omid E. David and Nathan S. Netanyahu. "DeepSign: Deep learning for automatic malware signature generation and classification." In: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280815.

[60] Gianni D'Angelo, Massimo Ficco, and Francesco Palmieri. "Malware detection in mobile environments based on Autoencoders and API-images." In: *Journal of Parallel and Distributed Computing* 137 (2020), pp. 26–33. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2019.11.001.

[61] Gianni D'Angelo, Francesco Palmieri, and Antonio Robustelli. "Effectiveness of Video-Classification in Android Malware Detection Through API-Streams and CNN-LSTM Autoencoders." In: *Mobile Internet Security*. Ed. by Ilsun You, Hwankuk Kim, Taek-Young Youn, Francesco Palmieri, and Igor Kotenko. Singapore: Springer Singapore, 2022, pp. 171–194. ISBN: 978-981-16-9576-6.

[62] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. "A Survey on Automated Dynamic Malware-Analysis Techniques and Tools." In: *ACM Comput. Surv.* 44.2 (2008). ISSN: 0360-0300. DOI: 10.1145/2089125.2089126.

[63] Ammar Elhadi, Mohd Maarof, and Bazara Barry. "Improving the Detection of Malware Behaviour Using Simplified Data Dependent API Call Graph." In: *International Journal*

of Security and Its Applications 7 (Sept. 2013), pp. 29–42. DOI: 10.14257/ijsia.2013.7.5.03.

[64] Yuxin Ding, Xiaoling Xia, Sheng Chen, and Ye Li. "A malware detection method based on family behavior graph." In: *Computers & Security* 73 (2018), pp. 73–86. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2017.10.007.

[65] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. "MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models (Extended Version)." In: *ACM Trans. Priv. Secur.* 22.2 (2019). ISSN: 2471-2566. DOI: 10.1145/3313391.

[66] Hyunjoo Kim, Jonghyun Kim, Youngsoo Kim, Ikkyun Kim, Kuinam J. Kim, and Hyuncheol Kim. "Improvement of Malware Detection and Classification Using API Call Sequence Alignment and Visualization." In: *Cluster Computing* 22.1 (2019), 921–929. ISSN: 1386-7857. DOI: 10.1007/s10586-017-1110-2.

[67] Gianni D'Angelo, Massimo Ficco, and Francesco Palmieri. "Association rule-based malware classification using common subsequences of API calls." In: *Applied Soft Computing* 105 (2021), p. 107234. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2021.107234.

[68] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data." In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1273–1282.

[69] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. "Federated Machine Learning: Concept and Applications." In: *ACM Trans. Intell. Syst. Technol.* 10.2 (2019). ISSN: 2157-6904. DOI: 10.1145/3298981.

[70] Ana Cholakoska, Bjarne Pfitzner, Hristijan Gjoreski, Valentin Rakovic, Bert Arnrich, and Marija Kalendar. "Differentially Private Federated Learningfor Anomaly Detection in EHealth Networks." In: *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*. New York, NY, USA: Association for Computing Machinery, 2021, 514–518. ISBN: 9781450384612.

[71] Maharshi Dhada, Amit Kumar Jain, and Ajith Kumar Parlikad. "Empirical Convergence Analysis of Federated Averaging for Failure Prognosis**This research was funded by the EPSRC and BT Prosperity Partnership project: Next Generation Converged Digital Infrastructure, grant number EP/R004935/1." In: *IFAC-PapersOnLine* 53.3 (2020). 4th IFAC Workshop on Advanced Maintenance Engineering, Services and Technologies - AMEST 2020, pp. 360–365. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.11.058.

[72] Ying Zhao, Junjun Chen, Di Wu, Jian Teng, and Shui Yu. "Multi-Task Network Anomaly Detection Using Federated Learning." In: *Proceedings of the Tenth International Symposium on Information and Communication Technology*. SoICT 2019. Hanoi, Ha Long Bay, Viet Nam: Association for Computing Machinery, 2019, 273–279. ISBN: 9781450372459. DOI: 10.1145/3368926.3369705.

[73] Tuo Zhang, Chaoyang He, Tianhao Ma, Lei Gao, Mark Ma, and Salman Avestimehr. "Federated Learning for Internet of Things." In: *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. SenSys '21. Coimbra, Portugal: Association for Computing Machinery, 2021, 413–419. ISBN: 9781450390972. DOI: 10.1145/3485730.3493444.

[74] Latif U. Khan, Walid Saad, Zhu Han, Ekram Hossain, and Choong Seon Hong. "Federated Learning for Internet of Things: Recent Advances, Taxonomy, and Open Challenges." In: *IEEE Communications Surveys Tutorials* 23.3 (2021), pp. 1759–1799. DOI: 10.1109/COMST.2021.3090430.

[75] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. "Federated Learning for Internet of Things: A Comprehensive Survey." In: *IEEE Communications Surveys Tutorials* 23.3 (2021), pp. 1622–1658. DOI: 10.1109/COMST.2021.3075439.

[76] Ruei-Hau Hsu, Yi-Cheng Wang, Chun-I Fan, Bo Sun, Tao Ban, Takeshi Takahashi, Ting-Wei Wu, and Shang-Wei Kao. "A Privacy-Preserving Federated Learning System for Android Malware Detection Based on Edge Computing." In: *2020 15th Asia Joint Conference on Information Security (AsiaJCIS)*. 2020, pp. 128–136. DOI: 10.1109/AsiaJCIS50894.2020.00031.

[77] Andrew Trask. *OpenMined/PySyft: A library for answering questions using data you cannot see*. 2017. URL: https://github.com/OpenMined/PySyft.

[78] Rafa Galvez, Veelasha Moonsamy, and Claudia Díaz. "Less is More: A privacy-respecting Android malware classifier using Federated Learning." In: *CoRR* abs/2007.08319 (2020). arXiv: 2007.08319. URL: https://arxiv.org/abs/2007.08319.

[79] Sanket Shukla, P D Sai Manoj, Gaurav Kolhe, and Setareh Rafatirad. "On-device Malware Detection using Performance-Aware and Robust Collaborative Learning." In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pp. 967–972. DOI: 10.1109/DAC18074.2021.9586330.

[80] Segun I. Popoola, Ruth Ande, Bamidele Adebisi, Guan Gui, Mohammad Hammoudeh, and Olamide Jogunola. "Federated Deep Learning for Zero-Day Botnet Attack Detection in IoT-Edge Devices." In: *IEEE Internet of Things Journal* 9.5 (2022), pp. 3930–3944. DOI: 10.1109/JIOT.2021.3100755.

[81] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders." In: *IEEE Pervasive Computing* 17.3 (2018), pp. 12–22. DOI: 10.1109/MPRV.2018.03367731.

[82] Reese Pathak and Martin J Wainwright. "FedSplit: an algorithmic framework for fast federated optimization." In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 7057–7066.

[83] Zachary Charles and Jakub Konečný. *On the Outsized Importance of Learning Rates in Local Update Methods*. 2020. DOI: 10.48550/ARXIV.2007.00878. URL: https://arxiv.org/abs/2007.00878.

[84] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. "Federated optimization in heterogeneous networks." In: *Proceedings of Machine Learning and Systems* 2 (2020), pp. 429–450.

[85] Yongqiang Lu, Zhaobin Liu, and Yiming Huang. "Parameters Compressed Mechanism in Federated Learning for Edge Computing." In: *2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. 2021, pp. 161–166. DOI: 10.1109/CSCloud-EdgeCom52276.2021.00038.

[86] Giovanni Paragliola and Antonio Coronato. "Definition of a novel federated learning approach to reduce communication costs." In: *Expert Systems with Applications* 189 (2022), p. 116109. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2021.116109.

[87] Rusul Abduljabbar, Hussein Dia, Sohani Liyanage, and Saeed Asadi Bagloee. "Applications of Artificial Intelligence in Transport: An Overview." In: *Sustainability* 11.1 (2019). ISSN: 2071-1050. DOI: 10.3390/su11010189. URL: https://www.mdpi.com/2071-1050/11/1/189.

[88] Aggeliki Androutsopoulou, Nikos Karacapilidis, Euripidis Loukis, and Yannis Charalabidis. "Transforming the communication between citizens and government through AI-guided chatbots." In: *Government Information Quarterly* 36.2 (2019), pp. 358–367. ISSN: 0740-624X. DOI: https://doi.org/10.1016/j.giq.2018.10.001.

[89] Christina L. McDowell Marinchak, Edward Forrest, and Bogdan Hoanca. "The impact of artificial intelligence and virtual personal assistants on marketing." In: *Encyclopedia of Information Science and Technology* (2018), pp. 5748 –5756. DOI: 10.4018/978-1-5225-2255-3.ch499.

[90] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[91] Mahmoud Abbasi, Amin Shahraki, and Amir Taherkordi. "Deep Learning for Network Traffic Monitoring and Analysis (NTMA): A Survey." In: *Computer Communications* 170 (2021), pp. 19–41. DOI: https://doi.org/10.1016/j.comcom.2021.01.021.

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

[93] Song Wang, Juan Fernando Balarezo, Sithamparanathan Kandeepan, Akram Al-Hourani, Karina Gomez Chavez, and Benjamin Rubinstein. "Machine Learning in Network Anomaly Detection: A Survey." In: *IEEE Access* 9 (2021), pp. 152379–152396. DOI: 10.1109/ACCESS.2021.3126834.

[94] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[95] Juan Antonio Morente-Molinera, József Mezei, Christer Carlsson, and Enrique Herrera-Viedma. "Improving Supervised Learning Classification Methods Using Multigranular Linguistic Modeling and Fuzzy Entropy." In: *IEEE Transactions on Fuzzy Systems* 25.5 (2017), pp. 1078–1089. DOI: 10.1109/TFUZZ.2016.2594275.

[96] Juan S. Angarita-Zapata, Antonio D. Masegosa, and Isaac Triguero. "A Taxonomy of Traffic Forecasting Regression Problems From a Supervised Learning Perspective." In: *IEEE Access* 7 (2019), pp. 68185–68205. DOI: 10.1109/ACCESS.2019.2917228.

[97] Xiaojin Jerry Zhu. "Semi-supervised learning literature survey." In: (2005).

[98] Xiaojin Zhu and Andrew B Goldberg. "Introduction to semi-supervised learning." In: *Synthesis lectures on artificial intelligence and machine learning* 3.1 (2009), pp. 1–130. DOI: https://doi.org/10.2200/S00196ED1V01Y200906AIM006).

[99] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[100] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[101] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. *Going Deeper with Convolutions*. 2014. DOI: 10.48550/ARXIV.1409.4842. URL: https://arxiv.org/abs/1409.4842.

[102] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. "Learning Hierarchical Features for Scene Labeling." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1915–1929. DOI: 10.1109/TPAMI.2012.231.

[103] Geoffrey Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups." In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. DOI: 10.1109/MSP.2012.2205597.

[104] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. "Natural Language Processing (Almost) from Scratch." In: *J. Mach. Learn. Res.* 12.null (2011), 2493–2537. ISSN: 1532-4435.

[105] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. DOI: 10.48550/ARXIV.1409.3215. URL: https://arxiv.org/abs/1409.3215.

[106] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. *On Using Very Large Target Vocabulary for Neural Machine Translation*. 2014. DOI: 10.48550/ARXIV.1412.2007. URL: https://arxiv.org/abs/1412.2007.

[107] Hui Y. Xiong et al. "The human splicing code reveals new insights into the genetic determinants of disease." In: *Science* 347.6218 (2015), p. 1254806. DOI: 10.1126/science.1254806. eprint: https://www.science.org/doi/pdf/10.1126/science.1254806. URL: https://www.science.org/doi/abs/10.1126/science.1254806.

[108] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. "Machine Learning and Deep Learning Methods for Cybersecurity." In: *IEEE Access* 6 (2018), pp. 35365–35381. DOI: 10.1109/ACCESS.2018.2836950.

[109] Anna L. Buczak and Erhan Guven. "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection." In: *IEEE Communications Surveys & Tutorials* 18.2 (2016), pp. 1153–1176. DOI: 10.1109/COMST.2015.2494502.

[110] Monika Roopak, Gui Yun Tian, and Jonathon Chambers. "Deep Learning Models for Cyber Security in IoT Networks." In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. 2019, pp. 0452–0457. DOI: 10.1109/CCWC.2019.8666588.

[111] Ankush Singla and Elisa Bertino. "How Deep Learning Is Making Information Security More Intelligent." In: *IEEE Security & Privacy* 17.3 (2019), pp. 56–65. DOI: 10.1109/MSEC.2019.2902347.

[112] D.W. Ruck, S.K. Rogers, M. Kabrisky, M.E. Oxley, and B.W. Suter. "The multilayer perceptron as an approximation to a Bayes optimal discriminant function." In: *IEEE Transactions on Neural Networks* 1.4 (1990), pp. 296–298. DOI: 10.1109/72.80266.

[113] John F. Kolen and Stefan C. Kremer. "Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies." In: *A Field Guide to Dynamical Recurrent Networks*. 2001, pp. 237–243. DOI: 10.1109/9780470544037.ch14.

[114] David E. Rumelhart and James L. McClelland. "Learning Internal Representations by Error Propagation." In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.

[115] Jeeheh Oh, Jiaxuan Wang, Shengpu Tang, Michael Sjoding, and Jenna Wiens. *Relaxed Parameter Sharing: Effectively Modeling Time-Varying Relationships in Clinical Time-Series*. 2019. DOI: 10.48550/ARXIV.1906.02898. URL: https://arxiv.org/abs/1906.02898.

[116] P.J. Werbos. "Backpropagation through time: what it does and how to do it." In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[117] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." In: *Neural Computation* 9.8 (1998), 1735–1780. DOI: https://doi.org/10.1162/neco.1997.9.8.1735.

[118] Alex Graves and Jürgen Schmidhuber. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks." In: *Proceedings of the 21st International Conference on Neural Information Processing Systems*. NIPS'08. Vancouver, British Columbia, Canada: Curran Associates Inc., 2008, 545–552. ISBN: 9781605609492.

[119] Alex Graves and Navdeep Jaitly. "Towards End-To-End Speech Recognition with Recurrent Neural Networks." In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, 2014, pp. 1764–1772. URL: https://proceedings.mlr.press/v32/graves14.html.

[120] Alex Graves. *Generating Sequences With Recurrent Neural Networks*. 2013. DOI: 10.48550/ARXIV.1308.0850. URL: https://arxiv.org/abs/1308.0850.

[121] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to Sequence Learning with Neural Networks." In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. Vol. 27. Curran Associates, Inc., 2014.

URL: https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

[122] Ryan Kiros, Ruslan Salakhutdinov, and Richard S. Zemel. *Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models*. 2014. DOI: 10.48550/ARXIV.1411.2539. URL: https://arxiv.org/abs/1411.2539.

[123] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

[124] Lan Huong Nguyen and Susan Holmes. "Ten quick tips for effective dimensionality reduction." In: *PLOS Computational Biology* 15.6 (June 2019), pp. 1–19. DOI: 10.1371/journal.pcbi.1006907. URL: https://doi.org/10.1371/journal.pcbi.1006907.

[125] Lingheng Meng, Shifei Ding, Nan Zhang, and Jian Zhang. "Research of stacked denoising sparse autoencoder." In: *Neural Computing and Applications* 30.7 (2018), pp. 2083–2100. DOI: 10.1007/s00521-016-2790-x. URL: https://doi.org/10.1007/s00521-016-2790-x.

[126] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. "Contractive Auto-Encoders: Explicit Invariance during Feature Extraction." In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, 833–840. ISBN: 9781450306195.

[127] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and Composing Robust Features with Denoising Autoencoders." In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, 1096–1103. ISBN: 9781605582054. DOI: 10.1145/1390156.1390294. URL: https://doi.org/10.1145/1390156.1390294.

[128] Jameson Thies and Amirhossein Alimohammad. "Compact and Low-Power Neural Spike Compression Using Undercomplete Autoencoders." In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 27.8 (2019), pp. 1529–1538. DOI: 10.1109/TNSRE.2019.2929081.

[129] Fanghua Ye, Chuan Chen, and Zibin Zheng. "Deep Autoencoder like Nonnegative Matrix Factorization for Community Detection." In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. CIKM '18. Torino, Italy: Association for Computing Machinery, 2018, 1393–1402. ISBN: 9781450360142. DOI: 10.1145/3269206.3271697. URL: https://doi.org/10.1145/3269206.3271697.

[130] Giannis Karamanolakis, Kevin Raji Cherian, Ananth Ravi Narayan, Jie Yuan, Da Tang, and Tony Jebara. "Item Recommendation with Variational Autoencoders and Heterogeneous Priors." In: *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*. DLRS 2018. Vancouver, BC, Canada: Association for Computing Machinery, 2018, 10–14. ISBN: 9781450366175. DOI: 10.1145/3270323.3270329. URL: https://doi.org/10.1145/3270323.3270329.

[131] Marco Maggipinto, Chiara Masiero, Alessandro Beghi, and Gian Antonio Susto. "A Convolutional Autoencoder Approach for Feature Extraction in Virtual Metrology." In: *Procedia Manufacturing* 17 (2018). 28th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2018), June 11-14, 2018, Columbus, OH, USAGlobal Integration of Intelligent Manufacturing and Smart Industry for Good of Humanity, pp. 126–133. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2018.10.023. URL: https://www.sciencedirect.com/science/article/pii/S2351978918311399.

[132] Milad Mohammadi and Subhasis Das. *SNN: Stacked Neural Networks*. 2016. DOI: 10.48550/ARXIV.1605.08512. URL: https://arxiv.org/abs/1605.08512.

[133] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning." In: *IEEE Transactions on Knowledge and Data*

Engineering 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[134] Muddasser Alam, Guifang Liu, Huaiqian Bao, and Baokun Han. "A Stacked Autoencoder-Based Deep Neural Network for Achieving Gearbox Fault Diagnosis." In: *Mathematical Problems in Engineering* 2018 (2018), p. 5105709. DOI: 10.1155/2018/5105709. URL: https://doi.org/10.1155/2018/5105709.

[135] Simon Washington, Matthew G. Karlaftis, and Fred Mannering. *Statistical and Econometric Methods for Transportation Data Analysis*. Chapman and Hall/CRC, 2003. DOI: https://doi.org/10.1201/9780429244018.

[136] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications, Springer Texts in Statistics*. Springer-Verlag, 2017. ISBN: 978-3-319-52451-1.

[137] M.b Priestley. *Nonlinear and Non-Stationary Time Series*. Academic Press, 1988.

[138] J. P. Eckmann and D. Ruelle. "Ergodic theory of chaos and strange attractors." In: *Rev. Mod. Phys.* 57 (3 1985), pp. 617–656. DOI: 10.1103/RevModPhys.57.617. URL: https://link.aps.org/doi/10.1103/RevModPhys.57.617.

[139] Floris Takens. "Detecting strange attractors in turbulence." In: 1981, 366–381. DOI: 10.1007/BFB0091924.

[140] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw. "Geometry from a Time Series." In: *Phys. Rev. Lett.* 45 (9 1980), pp. 712–716. DOI: 10.1103/PhysRevLett.45.712. URL: https://link.aps.org/doi/10.1103/PhysRevLett.45.712.

[141] Tim Sauer, James A. Yorke, and Martin Casdagli. "Embedology." In: *Journal of Statistical Physics* 3 (), pp. 579–616. DOI: 10.1007/BF01053745.

[142] Andrew M. Fraser and Harry L. Swinney. "Independent coordinates for strange attractors from mutual information." In: *Phys. Rev. A* 33 (2 1986), pp. 1134–1140. DOI: 10.1103/PhysRevA.33.1134. URL: https://link.aps.org/doi/10.1103/PhysRevA.33.1134.

[143] Matthew B. Kennel, Reggie Brown, and Henry D. I. Abarbanel. "Determining embedding dimension for phase-space reconstruction using a geometrical construction." In: *Phys. Rev. A* 45 (6 1992), pp. 3403–3411. DOI: 10.1103/PhysRevA.45.3403. URL: https://link.aps.org/doi/10.1103/PhysRevA.45.3403.

[144] Joseph P Zbilut, Alessandro Giuliani, and Charles L Webber. "Recurrence quantification analysis and principal components in the detection of short complex signals." In: *Physics Letters A* 237.3 (1998), pp. 131–135. ISSN: 0375-9601. DOI: https://doi.org/10.1016/S0375-9601(97)00843-8. URL: https://www.sciencedirect.com/science/article/pii/S0375960197008438.

[145] Don van Ravenzwaaij, Pete Cassey, and Scott D. Brown. "A simple introduction to Markov Chain Monte–Carlo sampling." In: *Psychonomic Bulletin & Review* 25.1 (2018), pp. 143–154. DOI: 10.3758/s13423-016-1015-8. URL: https://doi.org/10.3758/s13423-016-1015-8.

[146] L. Rabiner and B. Juang. "An introduction to hidden Markov models." In: *IEEE ASSP Magazine* 3.1 (1986), pp. 4–16. DOI: 10.1109/MASSP.1986.1165342.

[147] David Schön Myers, Lisa Wallin, and Petter Wikström. "An introduction to Markov chains and their applications within finance." In: *MVE220 Financial Risk: Reading Project* (2017), pp. 1–7.

[148] Joseph Rocca. *Introduction to Markov chains Definitions, properties, and PageRank example.* URL: https://towardsdatascience.com/brief-introduction-to-markov-chains-2c8cab9c98ab. (accessed: 24.02.2019).

[149] Francesco Palmieri. "Network anomaly detection based on logistic regression of nonlinear chaotic invariants." In: *Journal of Network and Computer Applications* 148 (2019), p. 102460. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2019.102460.

[150] Henri Poincaré. "Sur le problème des trois corps et les équations de la dynamique." In: *Acta mathematica* 13.1 (1890), A3–A270.

[151] Joseph P. Zbilut and Charles L. Webber. "Embeddings and delays as derived from quantification of recurrence plots." In: *Physics Letters A* 171.3 (1992), pp. 199–203. ISSN: 0375-9601. DOI: https://doi.org/10.1016/0375-9601(92)90426-M. URL: https://www.sciencedirect.com/science/article/pii/037596019290426M.

[152] Andrés F. Almeida-Ñauñay, Rosa M. Benito, Miguel Quemada, Juan C. Losada, and Ana M. Tarquis. "Recurrence plots for quantifying the vegetation indices dynamics in a semi-arid grassland." In: *Geoderma* 406 (2022), p. 115488. ISSN: 0016-7061. DOI: https://doi.org/10.1016/j.geoderma.2021.115488.

[153] Rainer Hegger, Holger Kantz, and Thomas Schreiber. "Practical implementation of nonlinear time series methods: The TISEAN package." In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 9.2 (1999), pp. 413–435.

[154] Alireza Makhzani and Brendan Frey. *k-Sparse Autoencoders*. 2014. arXiv: 1312.5663 [cs.LG].

[155] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization." In: *ICISSP*. 2018.

[156] Manu Mannattil. *NoLiTSA - NonLinear Time Series Analysis*. https://github.com/manu-mannattil/nolitsa. [Online; Last access 26-July-2022]. 2022.

[157] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques*. 4th. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016. ISBN: 0128042915.

[158] Wei Luo, Jun Li, Jian Yang, Wei Xu, and Jian Zhang. "Convolutional Sparse Autoencoders for Image Classification." In: *IEEE Transactions on Neural Networks and Learning Systems* 29.7 (2018), pp. 3289–3294. DOI: 10.1109/TNNLS.2017.2712793.

[159] Phillip Williams, Indira Kaylan Dutta, Hisham Daoud, and Magdy Bayoumi. "A survey on security in internet of things with a focus on the impact of emerging technologies." In: *Internet of Things* 19 (2022), p. 100564. ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2022.100564. URL: https://www.sciencedirect.com/science/article/pii/S2542660522000592.

[160] Chandra Shekhar Yadav, Jagendra Singh, Aruna Yadav, Himansu Sekhar Pattanayak, Ravindra Kumar, Arfat Ahmad Khan, Mohd Anul Haq, Ahmed Alhussen, and Sultan Alharby. "Malware Analysis in IoT & Android Systems with Defensive Mechanism." In: *Electronics* 11.15 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11152354. URL: https://www.mdpi.com/2079-9292/11/15/2354.

[161] Lakshmanan Nataraj, S. Karthikeyan, and B.S. Manjunath. "SATTVA: SpArsiTy Inspired ClassificaTion of Malware VAriants." In: *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security*. IH&MMSec '15. Portland, Oregon, USA: Association for Computing Machinery, 2015, 135–140. ISBN: 9781450335874. DOI: 10.1145/2756601.2756616.

[162] Nan Zhang, Kan Yuan, Muhammad Naveed, Xiao yong Zhou, and XiaoFeng Wang. "Leave Me Alone: App-Level Protection against Runtime Information Gathering on Android." In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2015, pp. 915–930. ISBN: 978-1-4673-6949-7.

[163] Hsin-Yu Chuang and Sheng-De Wang. "Machine learning based hybrid behavior models for Android malware analysis." In: *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE. 2015, pp. 201–206.

[164] Gernot A Fink. *Markov models for pattern recognition*. en. 2nd ed. Advances in Computer Vision and Pattern Recognition. London, England: Springer, Jan. 2014.

[165] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. "Communication-efficient learning of deep networks from decentralized

data." In: *Artificial intelligence and statistics* (2017), pp. 1273–1282.

[166] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. *Federated Learning with Matched Averaging*. 2020. DOI: 10.48550/ARXIV.2002.06440. URL: https://arxiv.org/abs/2002.06440.

[167] Sannara EK, Francois PORTET, Philippe LALANDA, and German VEGA. "A Federated Learning Aggregation Algorithm for Pervasive Computing: Evaluation and Comparison." In: *2021 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2021, pp. 1–10. DOI: 10.1109/percom50583.2021.9439129.

[168] Francois Carrez, Tarek Elsaleh, David Gómez, Luis Sánchez, Jorge Lanza, and Paul Grace. "A Reference Architecture for federating IoT infrastructures supporting semantic interoperability." In: *2017 European Conference on Networks and Communications (EuCNC)*. 2017, pp. 1–6. DOI: 10.1109/EuCNC.2017.7980765.

[169] Chi-Sheng Shih, Ching-Chi Chuang, and Hsin-Yuan Yeh. "Federating public and private intelligent services for IoT applications." In: *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2017, pp. 558–563. DOI: 10.1109/IWCMC.2017.7986346.

[170] Md Nazmus Sadat, Md Momin Al Aziz, Noman Mohammed, Feng Chen, Xiaoqian Jiang, and Shuang Wang. "SAFETY: Secure gwAs in Federated Environment through a hYbrid Solution." In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 16.1 (2019), pp. 93–102. DOI: 10.1109/TCBB.2018.2829760.

[171] Gianni D'Angelo, Francesco Palmieri, Antonio Robustelli, and Arcangelo Castiglione. "Effective classification of android malware families through dynamic features and neural networks." In: *Connection Science* 33.3 (2021), pp. 786–801. DOI: 10.1080/09540091.2021.1889977. eprint: https://doi.org/10.1080/09540091.2021.1889977.

[172] Cuckoo. *Automated malware analysis*. 2019. URL: https://cuckoosandbox.org/.

[173] De Jong Irmen. *Pyro - Python Remote Objects*. 2021. URL: https://pyro4.readthedocs.io/en/stable/#.

[174] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.