Mohammed VI Polytechnic University | College of Computing

Securing Networks:

Anomaly-Based Network Intrusion Detection with Federated Learning

Doctoral Thesis

submitted in fulfillment of the requirements for the degree of Doctor of Engineering (Dr.-Ing.)

by

Meryem Janati Idrissi, M.Sc.

Thesis Director : Dr. Ismail Berrada

College of Computing

Date of submission: 02.02.2024

Date of defense: 16.05.2024

Benguerir 2024

Affidavit

I, the undersigned, hereby declare that the submitted Doctoral Thesis is my own work. I have only used the sources indicated and have not made unauthorized use of the services of a third party. Where the work of others has been quoted or reproduced, the source is always given. I further declare that the dissertation presented here has not been submitted in the same or similar form to any other institution for the purpose of obtaining an academic degree.

Ben-Guerir, –.12.2023

Abstract

Over the last decades, the Internet has evolved into a complex infrastructure that connects billions of devices worldwide. This evolution has not only facilitated widespread connectivity but has also resulted in the broadening of the threat landscape. The increased interconnectivity of devices and systems has created new system vulnerabilities, placing cybersecurity at the forefront of concerns for organizations regardless of their scale or size. A robust cybersecurity strategy can no longer solely depend on static defenses of antiviruses and firewalls but demands layered protections. Intrusion Detection Systems (IDS) have emerged as a complementary security measure, demonstrating effectiveness in detecting various cyber threats. Centralized Machine Learning (ML)-based anomaly detection methods have shown promising results in improving the accuracy and efficiency of IDS. However, new challenges arise such as privacy concerns and regulatory restrictions that must be tackled. Federated Learning (FL) has emerged as a solution that allows distributed clients to collaboratively train a shared model while preserving the privacy of their local data. In this dissertation, we propose the use of ML/Deep Learning (DL) and FL to build efficient and privacy-preserving IDS.

Our first contribution proposes a distributed network intrusion detection method based on Autoencoders (AEs) and FL. Our approach, *Fed-ANIDS*, utilizes AEs to build an anomaly-based Network Intrusion Detection System (NIDS) to detect intrusions effectively while incurring low false positive rates. On the other hand, it provides privacy guarantees to protect sensitive client data by allowing each entity in the system to learn locally with its data. Evaluations of different datasets demonstrate the effectiveness of the proposed approach by achieving high performance in terms of different metrics while preserving the data privacy of distributed clients.

The second contribution of this thesis investigates the relationship between the quality of flow features and the performance of ML models. Specifically, by designing a series of experiments, we inspect the impact of the flow metering hyperparameters, specifically idle and active timeouts, on the overall performance of NIDS. Furthermore, we model a realistic scenario involving distributed and heterogeneous NIDS instances. For this purpose, we explore FL to extend our understanding of its potential in the field of heterogeneous NIDS. In light of our findings, fine-tuning idle and active timeouts is imperative.

The final contribution of this thesis proposes a novel solution, called *FedBS* for the data heterogeneity challenge in FL for particular ML models with batch normalization layers. *FedBS* consists of a new generic aggregation strategy that handles batch statistics in FL with non-Independent and Identically Distributed (IID) settings. Empirical results show that our proposed approach outperforms state-of-the-art methods on both various real-world datasets and under various non-IID settings. Moreover, in some cases, *FedBS* can be 2$\times$ faster than other FL approaches, coupled with higher testing accuracy.

Résumé

Au cours des dernières décennies, Internet a évolué en une infrastructure complexe qui relie des milliards d'appareils à travers le monde. Cette évolution a non seulement facilité la connectivité généralisée, mais a également élargi le paysage des menaces. L'interconnectivité croissante des appareils et des systèmes a créé de nouvelles vulnérabilités système, plaçant la cybersécurité au premier plan des préoccupations pour les organisations, quelle que soit leur taille ou leur envergure. Une stratégie de cybersécurité robuste ne peut plus dépendre uniquement des défenses statiques telles que les antivirus et les pare-feu, mais exige des protections en couches. Les systèmes de détection d'intrusion (IDS) ont émergé comme une mesure de sécurité complémentaire, démontrant leur efficacité dans la détection de diverses menaces cybernétiques. Les méthodes de détection d'anomalies basées sur l'apprentissage machine centralisé ont montré des résultats prometteurs pour améliorer la précision et l'efficacité des IDS. Cependant, de nouveaux défis surgissent, tels que les préoccupations en matière de confidentialité et les restrictions réglementaires, qui doivent être abordés. L'apprentissage fédéré (FL) est apparu comme une solution qui permet aux clients répartis de former collaborativement un modèle partagé tout en préservant la confidentialité de leurs données locales. Dans cette thèse, nous proposons l'utilisation de l'apprentissage machine (ML), l'apprentissage profond (DL) et du FL pour construire des IDS efficaces et respectueux de la vie privée.

Notre première contribution propose une méthode de détection d'intrusion dans les réseaux distribués basée sur les autoencodeurs (AEs) et le FL. Notre approche, *Fed-ANIDS* , utilise des AEs pour construire un système de détection d'intrusions réseau (NIDS) basé sur les anomalies afin de détecter efficacement les intrusions tout en générant des taux de faux positifs faibles. D'autre part, il fournit des garanties de confidentialité pour protéger les données sensibles des clients en permettant à chaque entité du système d'apprendre localement avec ses données. Les évaluations de différents ensembles de données démontrent l'efficacité de l'approche proposée en atteignant de hautes performances en termes de différents indicateurs tout en préservant la confidentialité des données des clients répartis.

La deuxième contribution de cette thèse examine la relation entre la qualité des caractéristiques de flux et la performance des modèles ML. Plus précisément, en concevant une série d'expériences, nous examinons l'impact des hyperparamètres de mesure de flux, notamment les temps d'attente inactifs et actifs, sur la performance globale des NIDS. De plus, nous modélisons un scénario réaliste impliquant des instances NIDS distribuées et hétérogènes. À cette fin, nous explorons le FL pour étendre notre compréhension de son potentiel dans le domaine des NIDS hétérogènes. À la lumière de nos résultats, l'ajustement fin des temps d'attente inactifs et actifs est impératif.

La dernière contribution de cette thèse propose une nouvelle solution, appelée *FedBS*, pour le défi de l'hétérogénéité des données dans le FL pour certains modèles ML avec des couches de normalisation par lots. *FedBS* se compose d'une nouvelle stratégie d'agrégation générique qui gère les statistiques de lots dans le FL avec des paramètres non indépendants et identiquement distribués (IID). Les résultats empiriques montrent que notre approche proposée surpasse les méthodes de pointe sur divers ensembles de données du monde réel et dans divers paramètres non IID. De plus, dans certains cas, *FedBS* peut être 2× plus rapide que d'autres approches FL, associé à une précision de test plus élevée.

Acknowledgments

This journey has been a rollercoaster, full of ups and downs, full of joy, stress, and excitement. First and foremost, all praise to Allah the Almighty and the Merciful for giving me the chance and strength to keep pushing forward, and to complete this thesis.

I would like to express my gratitude to my thesis supervisor Professor Ismail Berrada for his invaluable guidance, support, and mentorship throughout this research. His expertise, encouragement, and insightful feedback have been invaluable in shaping the direction and content of this thesis. Additionally, I extend my gratitude to all the professors and academic staff in the College of Computing for their encouragement and assistance throughout this academic journey.

Last but not least, I would like to express my sincere appreciation to my parents, my sister, and my brother for their support. They have always believed in me and have always been there for me, even during the most challenging times. I am grateful for their sacrifices and for everything they have done for me. And to my dear husband, I am forever indebted to you for being kind and supportive. Your assistance and encouragement were a constant source of strength, reminding me why I was striving for this goal. Finally, I extend my acknowledgment to my friends and colleagues for all the fun and happy moments we shared and for offering a sense of balance during this demanding journey.

Contents

List of Abbreviations

ML	Machine Learning
AI	Artificial Intelligence
FL	Federated Learning
DL	Deep Learning
NSS	Network Security Systems
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
NIDS	Network-based Intrusion Detection System
HIDS	Host-based Intrusion Detection System
AIDS	Anomaly-based Intrusion Detection System
SIDS	Signature-based Intrusion Detection System
DP	Differential Privacy
SMC	Secure-Multiparty Computation
HE	Homomorphic Encryption
NSM	Network Security Monitoring
ICS	Industrial Control Systems
SIEM	Security information and event management
SGD	Stochastic Gradient Descent
NN	Neural Network

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Network
RNN	Recurrent Neural Network
KNN	K-Nearest Neighbor
DT	Decision Tree
RF	Random Forest
LSTM	Long Short-Term Memory
ET	Extra Tree
MLP	Multi-Layer Perceptron
AE	Autoencoder
VAE	Variational Autoencoder
AAE	Adversarial Autoencoder
GAN	Genetative Adversarial Network
BiGAN	Bidirectional GAN
NF	Normalizing Flow
BN	Batch Normalization
IID	Independent and Identically Distributed
HFL	Horizontal Federated Learning
VFL	Vertical Federated Learning
FTL	Federated Transfer Learning

List of Figures

List of Tables

1 Introduction

Since the creation of the first computer virus *Creeper* in 1972 by Bob Thomas, attacks have increasingly become more complex and sophisticated, especially following the transition from ARPANET to the Internet in the 1980s. Since then, the Internet witnessed exponential growth in usage (Figure 1.1) and has become a defining characteristic of the digital age, driven by the widespread adoption of online services, the proliferation of connected devices, and the growing importance of digital communication. Everyday activities, once constrained by physical boundaries, are now effortlessly conducted in the virtual realm. From booking flights to paying bills, the Internet has become the invisible hand guiding our daily lives. We even seamlessly move substantial funds in the blink of an eye. The Internet's role as the virtual vault for our financial and personal data, while undeniably convenient, can't help but cause some unease in those who understand the inherent risks of this interconnected world. In fact, this surge in usage has created an expansive attack surface for cyber threats. As technology advances, so do the methods employed by malicious actors seeking to exploit vulnerabilities in the digital ecosystem [1]. Data breaches are alarmingly increasing in both magnitude and frequency, ransomware attacks are escalating at an unprecedented rate, and zero-day attacks are grabbing headlines. Meanwhile, traditional defenses such as firewalls and antiviruses are crumbling, and can no longer stand alone as the sole line of defense for networks in today's complex cyber threat landscape.

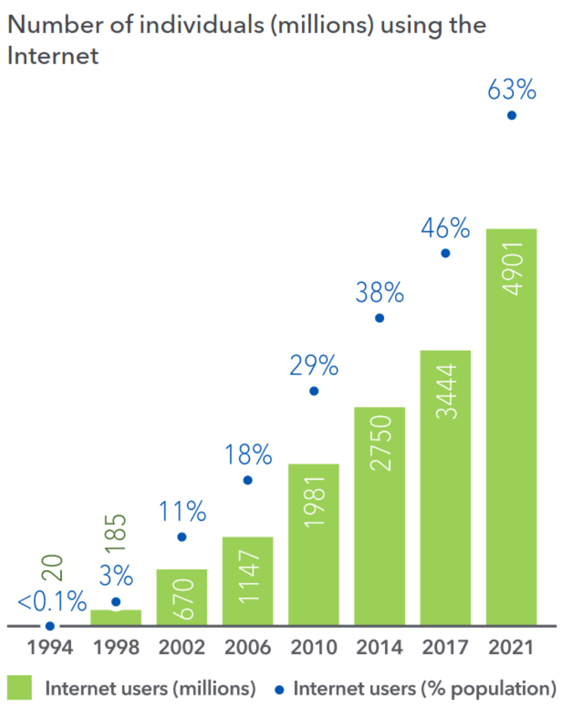


Figure 1.1: Growth of Internet use between 1994 and 2021 (source: ITU).

Within this context, network security has received a lot of attention from security researchers, as well as cybersecurity companies, striving to thwart cyberattacks. One of the most common Network Security Systems (NSS) used to secure networks is known as Intrusion Detection System (IDS). An IDS is a software or device that analyzes all traffic flowing through a network for potential intrusions and notifies the administrator upon detection so that they can be promptly addressed. While firewalls act as the first line of defense, effectively filtering out known threats, IDS provide an additional layer of security by actively monitoring network traffic for suspicious activity, thereby enhancing the overall cyber security posture. One of the key mottoes of information security by Eric Cole [2] is *"Prevention is Ideal, but Detection is a Must"*.

Decades of research have yielded a rich landscape of intrusion detection systems, giving rise to multifaceted IDS categorizations (see Chapter 2.3). Traditionally, IDS can be categorized based on several criteria, e.g., the analysis mode (real-time or offline) and the data processing architecture (centralized or distributed). The most fundamental classifications, however, lie in their detection technique and data source. When considering the detection method, two primary types of IDS exist: signature-based (SIDS) and anomaly-based (AIDS) systems. The former relies on a database of known signatures to identify attack attempts, while the latter identifies potential anomalies by detecting deviations from a pre-established baseline of normal traffic.

On the one hand, SIDS such as Snort [3] and Suricata [4], can quickly respond to known attacks with fewer false positives. However, they are unable to discover new attacks or previously unseen threats (zero-day attacks) and require frequent maintenance to update them. On the other hand, AIDS [5, 6, 7] are more adaptive and efficient in identifying new threats. This comes at the cost of being more vulnerable to false positives due to the inherent complexity of the techniques often employed for their implementation. Nevertheless, over the last two decades, extensive research efforts have been directed toward anomaly-based IDS. On the other hand, IDS are classified based on data sources into Host-based Intrusion Detection System (HIDS) and Network-based Intrusion Detection System (NIDS). HIDS is deployed on individual computers or servers, scrutinizing system logs, file integrity, and system calls to uncover suspicious activities specific to the host. In contrast, NIDS concentrates on monitoring network traffic for anomalies or patterns indicative of cyberattacks, utilizing sensors positioned strategically within the network infrastructure. While HIDS offers insights into host-specific activities, NIDS excels at detecting network-wide threats. According to this survey [8] conducted across 213 diverse organizations, each with at least 1,000 employees, Network Intrusion Detection/Prevention Systems (IDS/IPS) were heavily favored for network visibility securing the top position with a 92% preference. While host-based solutions secured the fourth position with a preference rate of 70% (see Figure 1.2).

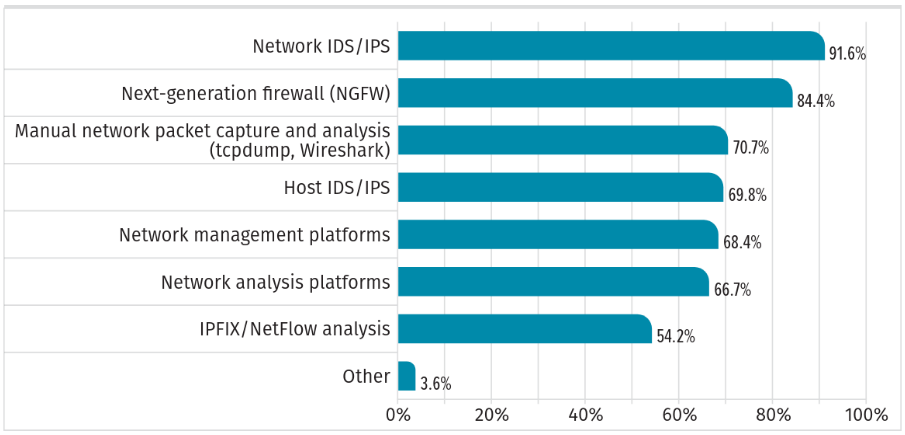


Figure 1.2: Tools in use by organizations for network visibility according to SANS 2020 Threat Detection Survey.

The world of NIDS, in turn, boils down to two main approaches, packet-based and flow-based, based on the source of data to be analyzed. Packet-based NIDS, also called, "traditional NIDS", inspect the whole content of individual data packets besides headers, providing a detailed view of network traffic. This in-depth analysis offers unparalleled precision, yet demands heavy resources, especially in high-traffic networks. Moreover, the increase in Internet usage has led to a significant uptick in Internet traffic. To meet the rising demand for data transmission, there have been substantial improvements in network infrastructure, resulting in notable enhancements in line speeds and bandwidth capacities. Nowadays, it is not uncommon to have access speeds ranging from 1 to 10 Gbps. Most universities, governmental institutions, and large corporations networks are moving from speeds in the hundreds of Mbps to speeds in the range of Gbps. Packet-based NIDS, however, are estimated to have a processing capacity ranging from 100 to 200 Mbps [9, 10]. Given these challenges, flow-based NIDS emerged as a promising candidate. Flow-based NIDS focus on the bigger picture [11], tracking and analyzing traffic flows rather than individual packets. A flow represents a sequence of packets between a specific source and destination over a specified time period. By offering a holistic view of the network behavior, this approach identifies suspicious patterns and trends within flows, while conserving resources.

To extract network flow features, several tools are available such as NFStream [12], CICFlowMeter[1], and nProbe[2]. These tools are vital for collecting and analyzing network flows, generally relying on a set of hyperparameters that configure and govern various aspects of the feature extraction process. These hyperparameters influence how features are selected, extracted, and represented from the raw network flow data, thus impacting the quality of the extracted features. One of these properties is the time interval, comprising two timeouts, an active and an idle (inactive) timeout [13]. The active timeout denotes the duration of an established connection during which data is actively transmitted between the source and the destination. In contrast, the idle timeout represents the duration of inactivity within an established connection. Modifying the values of these two parameters leads to the extraction of different features, potentially influencing the performance of ML models that rely on these features for their learning and decision-making processes.

1.1 Motivation

The phenomenal success of ML and DL techniques in areas such as image recognition and text classification has motivated security researchers to adopt these algorithms to IDS. Considering their potential to learn directly from the data, ML and DL algo-

[1] `https://github.com/ahlashkari/CICFlowMeter`
[2] `https://www.ntop.org/guides/nprobe/cli_options.html`

rithms are particularly beneficial for IDS given the diverse nature of network traffic. ML and DL algorithms can discern intricate patterns, anomalies, and subtle deviations from normal behavior in network traffic, making them particularly well-suited for the complex and evolving nature of cybersecurity challenges. However, through our research on ML and DL-based IDS, we've identified three areas that, we believe, need further investigation and improvement to address existing limitations and optimize the effectiveness of these algorithms in real-world cybersecurity scenarios.

First, despite the considerable number of ML and DL-based techniques proposed for IDS [14], the majority of them require centralizing all data for training purposes. Although this approach can provide high accuracy and performance in detecting threats, is not always feasible due to many constraints [15, 16]. Centralizing data may introduce a single point of failure, where the central server becomes a critical component, and any failure or compromise can impact the entire system. Let alone network latency and scalability issues, especially in large-scale networks. Moreover, beyond the operational challenges, the data gathered at the server may contain private sensitive users' information like personal details or financial transactions, which should not be publicly disclosed. Consequently, the adoption of centralized approaches can pose significant hurdles related to maintaining data confidentiality, integrity, and availability. The aggregation of sensitive information in one location increases the vulnerability to security breaches and unauthorized access. This is further exacerbated by the necessity to comply with strict privacy regulations, where mishandling or compromise of centralized data can result in legal consequences and reputational damage for the organizations.

Second, the quality of data features holds paramount importance for any ML/DL task. While sophisticated ML/DL algorithms fuel modern IDS, it's the raw data they feed on that truly determines their effectiveness. The performance of these algorithms, as intricate robots, is heavily dependent on the quality of the data they're given [17]. In the case of intrusion detection, selecting and extracting pertinent features from network traffic data is crucial for the overall performance and accuracy of these ML-based IDS solutions. Considered the building blocks of the algorithms, these features act as crucial indicators that aid in the identification of patterns associated with potential threats. Therefore, the meticulous process of feature extraction plays a critical role in enhancing the systems' capabilities to not only identify but also effectively mitigate emerging cyber threats. Moreover, the quality of features extracted directly impacts the algorithms' ability to adapt to evolving threat landscapes. In essence, the more accurate and pertinent the features, the more adeptly the ML-based IDS can recognize and respond to new and sophisticated forms of cyber attacks. Therefore, investing in refining and optimizing the feature extraction process becomes instrumental in fortifying the resilience and efficacy of IDS, ensuring they remain agile and effective in the face of dynamic cybersecurity challenges. Consequently, the feature extraction process significantly impacts these systems' capabilities to identify and effectively mitigate emerging threats.

Third, the integration of FL into applications, particularly IDS signifies a promising leap toward addressing the challenges associated with centralized methods. However, there are two sides to every coin, and this transformative approach is no exception. FL presents its own set of challenges, from communication bottleneck to poisoning attacks and scalability issues. However, one of the primary issues is the deployment of FL in heterogeneous systems, characterized by diversity in device capabilities and data distributions. Data distribution challenges in particular can hinder the convergence of the collaborative model. Diversity and variability in the quality and quantity of data across devices may impede the establishment of a cohesive global model. Specifically, if the data across different entities in FL is not IID, the learning in a FL setting becomes challenging. Most decentralized ML algorithms are envisioned with IID data in mind, implying that the datasets from different devices follow the same statistical distribution. However, in real-world situations, the data is commonly generated in different contexts, time windows, and locations. Thus, the IID assumption does not hold and the distribution of data stored across edge devices is not drawn from some global distribution and hence not representative of the overall distribution (non-IID data). In such scenarios, the model may struggle to generalize effectively, leading to suboptimal performance and potentially hindering the intended improvements in accuracy and efficiency.

1.2 Research questions and contributions

In light of what we have previously discussed, the goal of this thesis is to design efficient anomaly-based network intrusion detection systems, that can detect intrusions effectively while incurring low false positive rates. The latter could be accomplished by investigating the following directions: 1) Leveraging the FL approach to tackle the challenges of data privacy and scalability in NIDS. 2) Investigating the quality of data features on the overall performance of NIDS. To achieve these objectives, we will address the following research questions:

Research Question 1: What challenges do current centralized ML/DL-based NIDS approaches encounter in real-world applications? And what are the limitations of existing NIDS datasets?

Research Question 2: How can we mitigate the challenges of centralized NIDS approaches using FL?

Research Question 3: How does the quality of network data features affect the performance of NIDS?

Research Question 4: What are the new challenges created by using decentralized approaches for distributed systems?

Network intrusion detection has a long history of exploration, with numerous proposed methods over the years. To answer the **Research Question 1**, we start this thesis by studying existing works in the field and highlighting their challenges. Additionally, we discuss the drawbacks and limitations of benchmark network-based datasets and their impact on the development and performance of NIDS.

To answer **Research Question 2**, we propose a decentralized network intrusion detection method based on FL and autoencoders. The proposed method enables collaborative and secure learning of a global model for network intrusion detection by allowing each entity in the system to learn locally with its own data. To achieve this, we utilize autoencoders to build an anomaly-based system to detect possible attacks. Initially, each entity preprocesses its local data and trains the local model using only normal instances. Then, when all the entities complete the local training, the weights of local models are sent back to the server for aggregation. Finally, when the training is completed and convergence is reached, a threshold selection is performed to find the threshold value, the latter will be used in the detection phase. The main contributions of the paper are the following:

1. We propose, an autoencoder-based method for decentralized network intrusion detection systems by leveraging FL and anomaly detection. Furthermore, we study the impact of two FL methods, namely FedProx and FedAvg, on the model's performance in the context of NIDS.

2. We use more reliable flow features extracted from well-known NIDS datasets in an attempt to handle challenges related to flow construction, labeling, and attack simulation.

3. We conduct extensive evaluations using the various NIDS datasets. We also compare our method with Generative Adversarial Network-based models (GAN) that have been utilized for distributed NIDS in the literature. Moreover, we build and evaluate various scenarios to measure and investigate the generalization performance of our method with unseen datasets.

The purpose of **Research Question 3** is to shed light on the limitations of the existing state-of-the-art NIDS datasets. We address a particular facet of the issue at the level of network flow feature extraction. We provide an in-depth examination of the influence of flow timeouts (idle and active timeouts) values on the overall performance of ML models in detecting security threats. The main contributions are:

1. We conduct a thorough examination of the effects of different active and idle timeout values on the performance of ML models in the context of NIDS.
2. We assess the effectiveness of a variety of ML models, including ETC, RFC, and MLP models. This comprehensive evaluation offers insights into the suitability of various models for NIDS with various combinations of idle and active timeouts.
3. We assess the models' performance using distinct sets of features for each idle and active timeout couple. This analysis seeks to identify whether a specific flow timeout excels in performance.
4. We model a realistic scenario involving distributed NIDS instances, each owning data extracted using distinct idle and active timeouts. For this purpose, we explore FL and assess its performance to determine whether it introduces noteworthy enhancements. This investigation into FL extends our understanding of its potential in the field of heterogeneous NIDS.

The integration of FL into NIDS signifies a promising leap toward addressing the challenges associated with centralized methods. However, there are two sides to every coin, and this transformative approach is no exception. FL presents its own set of challenges. To answer **Research Question 4**, we discuss the traditional challenges of FL, particularly in the case of implementing FL for NIDS. We then propose a new method to handle the specific challenge of data heterogeneity for ML models with batch normalization layers. This work's main contributions are as follows:

1. We highlight the challenges of non-IID data and their impact on ML/DL models, especially those trained with batch normalization layers.

2. We propose a novel FL approach leveraging local batch statistics when using DL models with normalization layers in FL. The proposed method modifies the naive way of computing the weight vector of the local models based on the data size only.

3. We empirically demonstrate that our proposed approach outperforms both classical *FedAvg*, as well as the state-of-the-art *FedProx* through a comprehensive set of experiments conducted on various datasets under different non-IID data settings.

1.3 Thesis outline

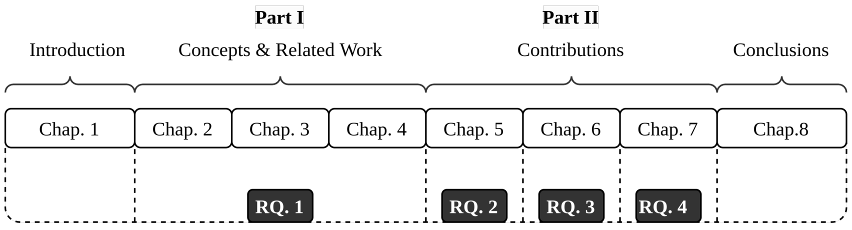


Figure 1.3: Thesis outline.

Figure 1.3 presents an overview of the structure of the dissertation. The remainder of this dissertation is structured as follows:

- **Part I** introduces the key concepts and information needed to navigate the depths of this dissertation. Furthermore, it discusses the existing body of work in both FL and IDS domains, establishing a comprehensive overview of the research field.
  - **Chapter 2** introduces intrusion detection systems, its definition, categorizations, and datasets.
  - **Chapter 3** gives a ML introduction including the different paradigms and tasks of ML, as well as various models and evaluation metrics. Additionally, it delved into the subfield DL and the decentralized framework FL.
  - **Chapter 4** discusses the application of ML/DL models and FL methods for intrusion detection and discusses existing work in the literature.

- **Part II** presents our contributions in both fields of intrusion detection and FL.
  - **Chapter 5** presents our autoencoder-based distributed learning framework for NIDS, called Fed-ANIDS. It describes its different components and discusses the evaluation results.
  - **Chapter 6** introduces the different components of the experimental design to study the impact of flow timeouts on the performance of NIDS. Then it describes the implementation details and the experimental results and findings.
  - **Chapter 7** highlights the different challenges of the proposed methods and proposes a novel method to address the specific challenge of data heterogeneity.
- **Chapter 8** concludes the thesis by drawing conclusions, findings, and future works.

1.4 Publications

- FEDBS: Learning on Non-IID Data in Federated Learning using Batch Normalization. Meryem Janati Idrissi, Ismail Berrada and Guevara Noubir. In 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI), pp. 861-867, 2021.
- Fed-ANIDS: Federated learning for anomaly-based network intrusion detection systems. Meryem Janati Idrissi, Hamza Alami, Abdelkader El Mahdaouy, Abdellah El Mekki, Soufiane Oualil, Zakaria Yartaoui and Ismail Berrada. In Expert Systems with Applications, vol. 234, no. , pp. 0957-4174, 2023.
- Flow timeout matters: Investigating the impact of active and idle timeouts on the performance of machine learning models in detecting security threats. Meryem Janati Idrissi, Hamza Alami, Abdelkader El Mahdaouy, Abdelhak Bouayad, Zakaria Yartaoui, and Ismail Berrada. Submitted to Future Generation Computer Systems Journal, Elsevier.

Part I

Concepts & Related Work

Part I serves as an introduction to the key concepts and technologies employed in the scope of this thesis. This part is divided into three main chapters. The first chapter defines intrusion detection, reviews the different types of IDS, and discusses various network-based datasets. The second chapter provides a detailed introduction to ML, DL, and FL. The last chapter discusses ML, DL, and FL for anomaly-based intrusion detection.

2 Intrusion Detection Systems

2.1 Introduction

Confidentiality, integrity, and availability, collectively known as the C.I.A triad (Figure 2.1), constitute three essential pillars of information security. An *"intrusion"* refers to any unauthorized or malicious activity that undermines one, both, or all of these integral components within an information system. It encompasses actions that violate the established security policies and threaten the security of a computer system, network, or application. Intrusions can take various forms, including unauthorized access attempts, exploitation of vulnerabilities, injection of malicious code, or any other activity that could potentially lead to a security breach.

- Confidentiality: It ensures that information is accessible only to authorized individuals, safeguarding against unauthorized access and disclosure. This is often achieved through encryption, access controls, and user authentication mechanisms. Data classification plays a crucial role in categorizing information based on its sensitivity and applying appropriate access restrictions.

- Integrity: It focuses on maintaining the accuracy and trustworthiness of data by guarding against unauthorized alterations. Hash functions and checksums are employed to verify data integrity. Digital signatures provide a way to authenticate the source and integrity of messages or documents. Access controls and version control systems contribute to maintaining data integrity.

- Availability: It ensures that information and resources are consistently accessible and usable by authorized users when needed. Redundancy and fault-tolerant systems are implemented to mitigate the impact of hardware failures or other issues. Disaster recovery planning and regular system monitoring contribute to ensuring the continuous availability of critical systems.

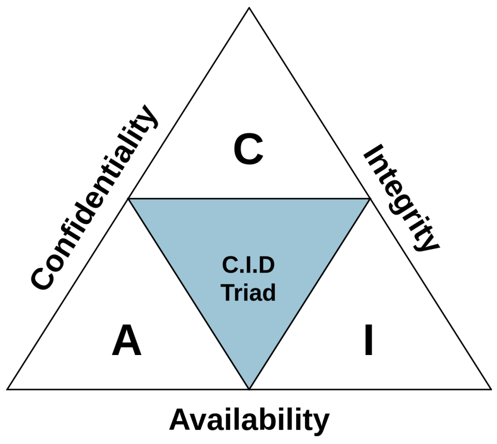


Figure 2.1: The C.I.A triad.

The process of monitoring and analyzing network or system activities to identify and respond to suspicious or malicious behavior is identified as *"intrusion detection"*. Any software or hardware designed to perform intrusion detection is categorized as an IDS.

2.2 Definition

An IDS acts as a digital guardian, that serves as a vital component of cybersecurity, providing an additional layer of defense against potential threats. IDS operates by examining network traffic or system logs for patterns indicative of unauthorized access, attacks, or abnormal activities. They detect and deal with insider attacks, as well as, external attacks. Upon detection, the IDS generates alerts and notifies administrators or takes predefined actions to mitigate the threat, for instance, collecting the alarms through the use of a Security Information and Event Management (SIEM) system. A SIEM system offers immediate analysis of outputs from various sources, correlating diverse alerts to present a comprehensive perspective on IT security.

IDS are at times confused with other security tools, such as firewalls and Intrusion Prevention Systems (IPS). Despite their shared objective of safeguarding systems within a network, these security mechanisms operate through distinct means. Firewalls act as a protective barrier between trusted internal networks and untrusted external networks, regulating incoming and outgoing network traffic based on predefined security rules. They analyze packets' headers to filter the traffic based on IP address, protocol, port number, etc. Firewalls should be placed as the first line of defense. On the other hand, Intrusion Prevention System (IPS) go beyond the capabilities of IDS by not only detecting but also actively blocking or preventing identified malicious activities. While IDS emphasize detection and alerting, IPS take immediate action to thwart potential threats, contributing to a more proactive security posture. Recognizing the nuanced roles of IDS, firewalls, and IPS is imperative for organizations to implement a comprehensive security strategy that effectively addresses various aspects of network protection.

2.3 Types of intrusion detection systems

IDS can be categorized into several types based on many characteristics as illustrated in Figure 2.2, each serving specific security needs within a networked environment. One of these characteristics is the deployment method which defines the location of the IDS and the type of activities that are inspected. Therefore, an IDS could be Host-based (HIDS), Network-based (NIDS), or Application-based IDS (AppIDS).

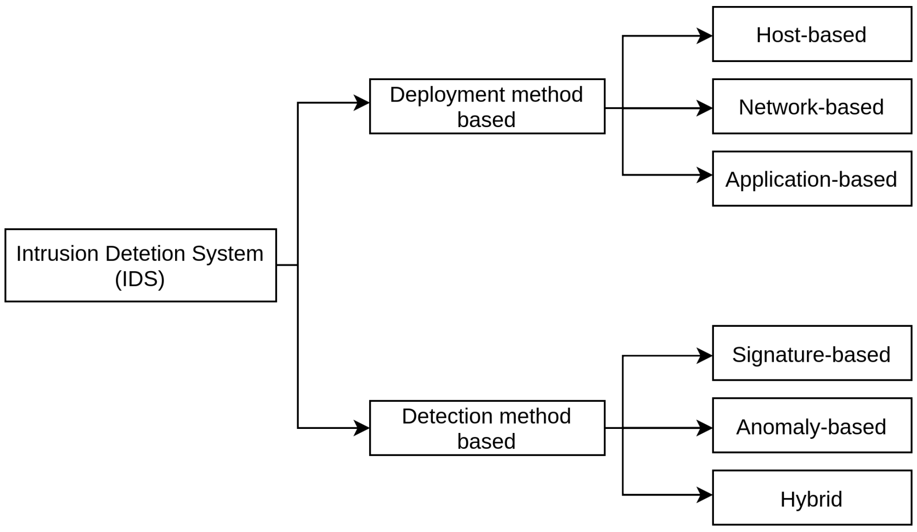


Figure 2.2: Taxonomy of intrusion detection systems.

- Host-based IDS [18] are installed on individual hosts or devices, closely monitoring activities like file modifications, login attempts, or system calls. HIDS analyzes host-specific behaviors, comparing them to predefined rules or baselines to identify any deviations that may signify intrusions or anomalies at the host level.

- Network-based IDS [19] are strategically deployed at key points in the network infrastructure, such as routers or switches, to analyze real-time traffic and detect patterns indicative of known attacks or anomalies. NIDS primarily focus on monitoring network-level activities, providing insights into unauthorized access attempts, malware presence, or unusual data transmission patterns.

- Application-based IDS [20] are a specialized type of HIDS designed to monitor and protect specific applications within a computing environment. Unlike network-based or host-based IDS that operate at broader levels, Application-based IDS focuses specifically on the security of individual software applications.

IDS are also categorized based on their detection methods, which describe how the detection engine works. Signature-based Intrusion Detection System (SIDS), Anomaly-based Intrusion Detection System (AIDS), and Hybrid detection are three primary types.

- Signature-based IDS, also known as misuse detection, operate on the fundamental principle of recognizing known attack patterns through a comprehensive database of attack signatures. This type of IDS scrutinizes incoming network traffic or system activity, comparing observed data against a repository of predefined signatures that represent distinct characteristics of previously identified threats. These signatures can manifest in various forms, ranging from simple sequences of bytes or characters to more complex representations such as branching tree diagrams. The IDS essentially functions as a vigilant gatekeeper, scanning the input stream for any matches with the stored signatures. Similar to the methodology employed by traditional antivirus software, signature-based IDS excels at the precise identification of known threats, ensuring a high level of specificity in threat detection. However, its effectiveness is inherently constrained by its reliance on a static signature database, making it less adaptive to emerging or novel threats (zero-day attacks). Regular and timely updates to the signature database are imperative to enhance the IDS's capability to recognize the latest attack patterns.

- Anomaly-based IDS employ a distinct approach to safeguard computer networks and systems by focusing on deviations from established patterns of normal behavior. Instead of relying on a predefined database of attack signatures, this type of IDS creates a baseline of expected or typical activities within the network or system. It continuously monitors and learns the usual patterns, generating alerts or responses when observed behavior deviates from this baseline. AIDS is particularly adept at identifying novel or previously unseen attacks, as it doesn't rely on predetermined signatures for detection. This adaptability makes it a valuable asset in addressing emerging threats that may not conform to known attack patterns. The system analyzes various parameters, including network traffic, user behavior, or system processes, to discern anomalies that might indicate a potential security breach. However, AIDS is more vulnerable to false positives due to the inherent complexity of the techniques often employed for their implementation. Different techniques are used for AIDS. Statistical-based, data mining-based, knowledge-based, and ML-based anomaly detection are the most used techniques. The complete taxonomy of AIDS is illustrated in Figure 2.3. Section 4.2 and 4.3 discuss the ML and DL-based techniques, which are the focus of this thesis.

- Hybrid detection represents a sophisticated approach that combines the strengths of both Signature-based and Anomaly-based methodologies to provide a comprehensive and adaptive security solution. In a hybrid model, the IDS leverages a database of known attack signatures, similar to the Signature-based approach, allowing for precise identification of well-defined threats based on historical knowledge. Simultaneously, it incorporates anomaly-based detection, creating a baseline of normal behavior to identify deviations that might signify novel or previously unseen attacks. This dual-layered strategy aims to enhance the overall efficacy of intrusion detection by offering both specificity for known threats and adaptability to emerging or sophisticated attacks.

2.4 Flow-based network anomaly detection

2.4.1 IP flows

Traditionally, NIDS employ Deep Packet Inspection (DPI) [21] which involves a thorough analysis of the data packets traveling through a network. Each packet contains information about its source, destination, content, and other relevant details. DPI

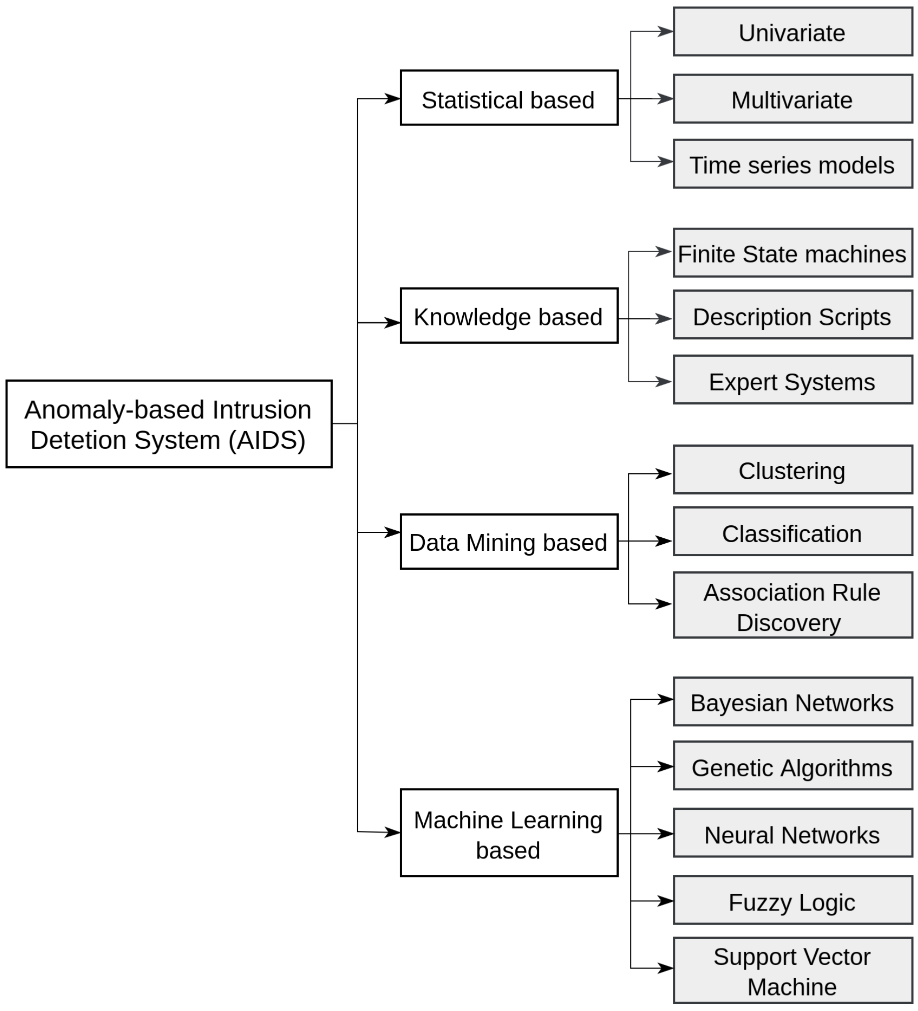


Figure 2.3: Taxonomy of anomaly-based intrusion detection system.

goes beyond merely examining the headers of these packets; it delves deep into the packet's payload, inspecting the actual data within. By inspecting the contents of data packets, payload-based NIDS can identify known attack signatures, unusual data patterns, or deviations from normal network behavior. However, inspecting the entire payload incurs significant computational overhead and hinders performance, especially in high-speed IP networks [22]. Moreover, the proliferation of encrypted protocols in network traffic adds substantial complexity to the task of implementing packet-based NIDS [23].

With these challenges in mind, researchers have shifted their attention toward flow-based methods [11]. Figure 2.4 shows the evolution of payload-based and flow-based intrusion detection. Flow-based NIDS leverage network flow records as their input source to detect potentially malicious activities. In the context of computer networks, a flow refers to a sequence of packets that share certain common attributes and characteristics as they traverse a network. In contrast to payload-based NIDS, flow-based NIDS deal with considerably reduced amounts of data as only the flow records are analyzed. In other words, with such systems, the aggregated information of the network is inspected instead of the content of data packets. According to a study that was conducted on the University of Twente (UT) network [13], the analysis of flow packets within a network typically accounts for approximately 0.1% of the overall network traffic. For network load, the added overhead resulting from flow collection and export protocol (NetFlow) averages around 0.2%. Another advantage of flow-based NIDS is the ease with which flow data can be collected from network devices that employ standard and widely recognized protocols such as Cisco, NetFlow, and IETF IPFIX, without the need for additional software installations. Flow-based NIDS are therefore a rational choice for high-speed networks.

The IP Flow Information Export (IPFIX) proposed a standard definition of network flow [24]:

> *"A Flow is defined as a set of packets or frames passing an observation Point in the network during a certain time interval. All packets belonging to a particular Flow have a set of common properties."*

These properties are called *flow keys* and typically are a 5-tuple consisting of the source and destination IPs and ports and the protocol:

```
(ip_src, ip_dst, port_src, port_dst, proto)
```

The typical structure of a network flow monitoring tool consists of three main steps: *packet observation*, *flow metering and export*, and *flow collection*. First, the packet observation stage is designed to capture packets from an *observation point* and pre-process them. Next, the metering process generates flow records based on headers extracted from packets collected from the observation point. Each header is then marked with a timestamp. Every incoming packet header initiates an update to an existing flow entry in the flow cache used to temporarily store flow entries and maintain a record of active flows in real-time. If the packet header matches an existing entry then the flow features are updated. Otherwise, a new flow entry is initiated. A flow record is forwarded to the collecting process once it has expired. For both NetFlow [25] and IPFIX [26], the metering process terminates a flow record for the following reasons:

- *Active timeout*: Flows that remain active beyond this specified time duration are considered expired and the corresponding flow records are sent to the flow collector. Common timeout values typically fall within the range of 2 minutes to 30 minutes (NetFlow).

- *Idle timeout*: If no packets have been observed in the flow for a longer time than this value then the flow is expired. Common timeout durations vary from 15 seconds (NetFlow) to 5 minutes.
- *Natural expiration*: The detection of a Transmission Control Protocol (TCP) packet with either the FIN or RST flag set indicates the termination of the TCP connection.
- *Emergency expiration*: If the flow cache memory is exhausted, a subset of flow entries are immediately terminated and exported to the collector.

Finally, the flow collector stores flow records in a format suitable for subsequent monitoring or analysis.

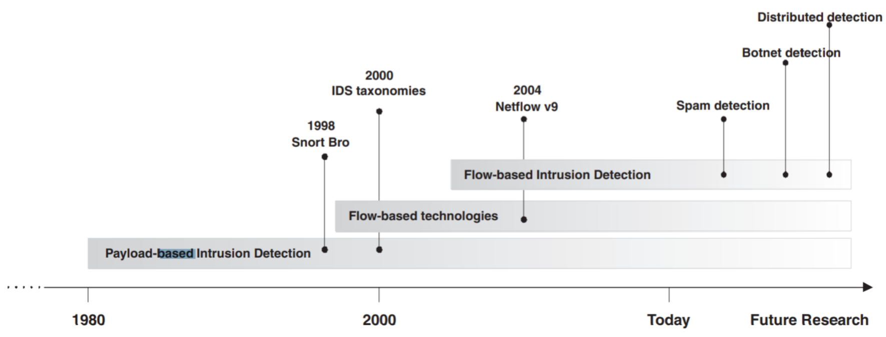


Figure 2.4: The evolution of intrusion detection and flow-based technologies (source: (Sperotto et al., 2010)).

2.4.2 Active and idle timeouts

In the realm of networking, *active* and *idle* timeouts are two parameters that are commonly used in various network protocols and devices to manage flows (network connections and sessions). Active timeout, sometimes termed connection timeout, places a predetermined limit on the maximum duration a flow can remain open, irrespective of data activity. It measures the time between successive packets or data exchanges. If no data is transmitted within the specified period, the flow may be considered inactive or stalled, and the active timeout timer will start counting down. Once the active timeout expires, the flow is terminated. The principal purpose of active timeout is to prevent flows from persisting indefinitely, thereby conserving resources and ensuring network health. Active timeouts are fundamental to network protocols like TCP, contributing significantly to the maintenance of flow integrity and the prevention of resource exhaustion.

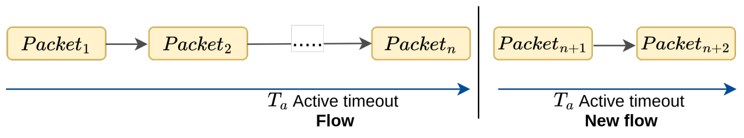


Figure 2.5: Active timeout.

Idle timeout, also referred to as inactive timeout, is the period of inactivity between a source and a destination before the flow is considered idle and subsequently terminated. It measures the duration during which there is no data transmission occurring between the two endpoints. When the idle timeout expires, the flow is closed, and the resources associated with that connection are freed up for other tasks. This helps prevent flows from being kept open indefinitely when there is no ongoing communication. Therefore, idle timeout plays a central role in minimizing security risks by closing idle connections as well as optimizing the allocation of incoming network traffic. Figures 2.5 and 2.6 provide illustrative representations of the active and idle timeouts respectively.

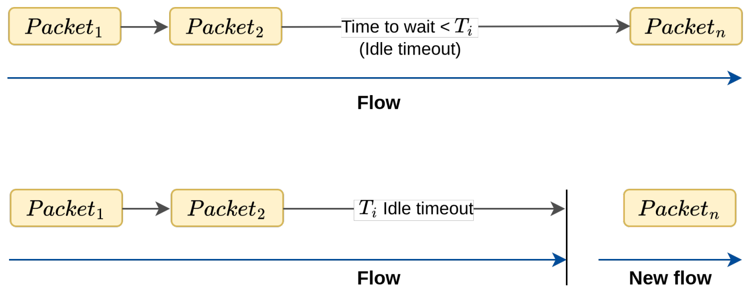


Figure 2.6: Idle timeout.

The values of idle and active timeouts in network flow analysis hold a substantial influence over the features extracted from the data, and consequently, may impact the performance of ML models optimized with these features. These timeout parameters influence the granularity and relevance of the extracted features. For instance, shorter idle and active timeouts can yield finer-grained features that capture transient or rapid changes in network activity, while longer timeouts tend to result in coarser features representing broader time intervals. Moreover, these choices bear on the level of noise reduction within the data; shorter timeouts may help filter out idle or sporadic connections but risk discarding benign ones, whereas longer timeouts could introduce more noise. Furthermore, the sensitivity of the model to network dynamics and its computational resource utilization are both affected by the choice of timeout values. Striking the right balance between capturing fine details and managing resources is essential for optimal model performance, as is ensuring that the selected timeout values align with the specific network environment and analytical goals, ultimately influencing the model's ability to detect patterns and anomalies effectively.

2.5 Network-based intrusion detection datasets

Network-based datasets play a crucial role in both the development and validation of NIDS [27]. They enable the comparison of various NIDS solutions, assisting security professionals in selecting the most suitable options for their specific environments. In the following paragraphs, we will discuss some of the key and most up-to-date NIDS datasets used within the scope of this thesis. We will also highlight some of the limitations associated with these datasets addressed in the literature.

USTC-TFC2016 [28] is a prominent dataset that primarily contains two parts. The first part is composed of 10 varieties of malware traffic collected from public websites hosted in a real network environment within the time period 2011 to 2015. The second part contains ten genres of normal traffic collected using IXIA BPS [1]. This dataset comes in *PCAP* format with a total size of 3.71 GB.

CIC-IDS2017 [29] is a state-of-the-art network intrusion detection dataset created by the Canadian Institute for Cybersecurity (CIC). It provides real-world network traffic data ("Benign" and "Attacks") in the *PCAP* format collected during the period of five days (Monday through Friday) in a controlled environment, aiming to simulate realistic network scenarios for analyzing and developing intrusion detection techniques.

CSE-CIC-IDS2018 is an improved version of the renowned *CIC-IDS2017* dataset. It was generated and launched by The Canadian Institute for Cybersecurity in 2018/2019[2]. It represents a significant improvement over its predecessor, incorporating up-to-date attack types and more comprehensive network intrusion scenarios. CSE-CIC-IDS2018 consists of seven up-to-date attack types including Botnet, Brute-

[1]`https://www.ixiacom.com/products/breakpoints`
[2]`https://www.unb.ca/cic/datasets/ids-2018.html`

force, Denial of Service and Distributed Denial of Service, Heartbleed, Inside Network Infiltration, and Web Attacks.

UNSW-NB15 [30] is a modern KDD-19 alternative created and released in 2015. The IXIA PerfectStorm tool was used to create a hybrid testbed-based normal and abnormal network traffic. 87.35% of the data is benign flows while the 12.65% left is attack flows. The dataset consists of nine attack classes, namely, Fuzzers, Reconnaissance, DoS, Backdoors, Generic, Analysis, Worms, Shellcode, and Exploits. Note that the traffic was captured in the *PCAP* format.

CUPID [31] [3] is a recent dataset that was created specifically for evaluating NIDS. CUPID was annotated with penetration testing to reflect both automated and human-generated attacks. It provides diverse attack types, including, Webcrawling, ARP, nmap, recorded live user interaction, DNSMap, Dig, Nslookup, DNSTracer, Password Brute Forcing, SQLi, Directory Traversal, DHCP attacks, STP, and Delivery of Reverse Meterpreter Shell. The dataset was captured in the *PCAP* format and processed using CICFlowMeter comprising approximately 50 GB.

ML algorithms thrive on data, both in quantity and quality. Adequate data is essential for the training and evaluation of ML algorithms, playing a pivotal role in the success of any ML task including intrusion detection. The quality of NIDS datasets is therefore of paramount importance for reliable and accurate systems. Nevertheless, because of the challenges involved in acquiring realistic labeled network data flows, researchers have developed benchmark NIDS datasets by simulating network behaviors in controlled testbed environments [27]. Using appropriate tools and frameworks, network flows are generated through several phases including data collection, labeling, and feature extraction. Nevertheless, multiple investigations revealed that NIDS datasets exhibit various deficiencies within one or more of these phases. In the following, we discuss works that analyzed NIDS datasets and pinpointed some of their shortcomings and flaws. This survey [32] on network threats and NIDS datasets raised a valid concern regarding the limitations of NIDS datasets. Current datasets in use for NIDS are often outdated and cover only 33.3% of known attacks. The paper also pointed out that the existing NIDS datasets are not representative enough of real-world traffic and include a limited variety of network attack types. Therefore, the current dataset might not be sufficient to keep up with the ever-evolving and complex threat landscape in addition to their effect on the building of NIDS. Likewise,

[3]The Colorado University Pentesting Intrusion Dataset (CUPID) `https://cupid.directory/`

Sarhan et al. [33] highlighted some limitations of existing NIDS datasets, particularly, the lack of a common ground feature set shared among datasets. This makes it difficult to fairly evaluate ML models' performance and their generalization ability across different datasets. To overcome this limitation, the authors proposed to use a standardized feature set that can be used by researchers to train and cross-evaluate various NIDS models. Engelin et al. [34] investigated the widely-used dataset CIC-IDS2017 and identified some major flaws related to flow construction, feature extraction, and labeling process. In their attempt to fix these issues, the authors released an update of CICFlowMeter. CSE-CIC-IDS2018 dataset [4] which is considered a reliable and updated version of CIC-IDS2017 was also found to be flawed [35] through a series of experiments. These issues are mainly related to feature generation and labeling.

The studies and analyses discussed reveal that the publically available NIDS datasets have various issues that can significantly impact the quality and performance of NIDS. As NIDS rely on these datasets for training and evaluation, the presence of flaws can hinder their ability to accurately detect and respond to security threats. Therefore, identifying and addressing these dataset limitations and enhancing their quality is crucial for the development and improvement of NIDS.

2.6 Summary

This chapter gave essential background information on intrusion detection. It commenced by offering a clear definition of an IDS and highlighting its pivotal role in cybersecurity. Moving forward, the chapter explored the diverse taxonomies of IDS based on two key characteristics, deployment method, and detection method. The details of these taxonomies were thoroughly examined to provide a comprehensive understanding. We then discussed the main two methods for network intrusion detection, namely, payload-based and flow-based network intrusion detection, emphasizing the transition from payload-based to flow-based methodologies and underscoring the evolving landscape and contemporary trends in intrusion detection strategies. In the same section, we defined IP flows within the context of network monitoring and we sketched the main phases of the flow creation process with a particular emphasis on active and idle timeouts. Finally, we presented various well-known and publically available IDS datasets that were used within the scope of this thesis.

[4]`https://www.unb.ca/cic/datasets/ids-2018.html`

The following chapter will present the key concept of ML and its subfields DL and FL.

3 Machine Learning Introduction

3.1 Introduction

Artificial Intelligence (AI) stands out as one of the most revolutionary technologies in the 21st century, fostering significant advancements across diverse industries, reshaping the employment landscape, and transforming the way societies function. This transformative field encompasses a spectrum of disciplines such as computer science, mathematics, and philosophy, aiming to create intelligent systems that emulate human-like cognitive functions. At its core, AI seeks to develop machines capable of learning, reasoning, problem-solving, and adapting to diverse scenarios. Unlike traditional tools confined to pre-defined rules and routines, AI delves into the vast ocean of data, unearthing hidden patterns and insights invisible to human eyes. This ability to learn from experience, adapt to evolving scenarios, and make predictions based on complex relationships transcends what any previous technology could achieve. ML, a sub-discipline of AI, has emerged as a key component in the evolution of this field. The availability of vast datasets and advancements in computing power have fueled the explosion of ML techniques. One of the most intriguing developments in ML is the emergence of DL, a subset of ML that mimics the human brain's neural networks. DL revolutionized AI, achieving remarkable breakthroughs in areas such as image recognition, natural language processing, and robotics.

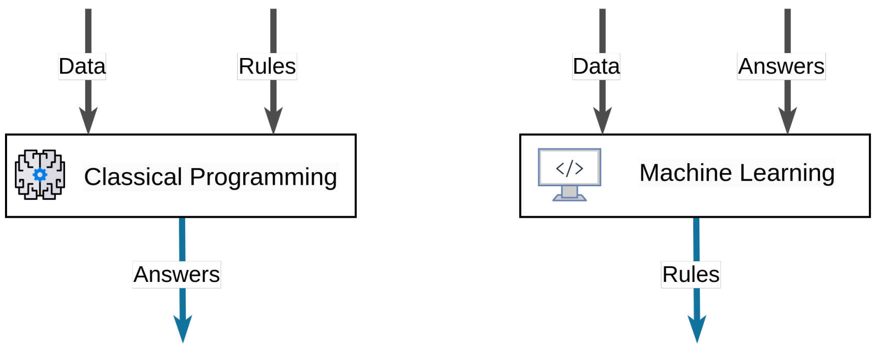


Figure 3.1: Classical programming vs machine learning.

3.2 What is machine learning?

ML is a subfield of AI, focused on the development and application of algorithms that allow computers to learn patterns and make predictions or take actions based on past experiences without being explicitly programmed [36]. Unlike traditional programming, where rules are explicitly defined, ML models learn by analyzing data (Figure 3.1). This data can be anything from text and images to numbers and sensor readings. Through various algorithms, the models discover patterns and relationships within the data, such as correlations between features or specific sequences that indicate certain events. Once trained, the models can then use the learned patterns to make predictions on new, unseen data. This could involve tasks like classifying emails as spam or not spam, recognizing objects in images, or even generating new creative content like music or poems. ML has attracted increasing attention from both academia and industry and is being applied in a wide range of fields such as healthcare, finance, recommendation systems, and network traffic analysis. At its core, ML follows a dual-phase approach: (i) First, in the training phase, the model is fed a large and diverse dataset of examples called the "training data". Algorithms come into play during this phase. They analyze the training data, uncovering patterns and relationships between features. Based on these patterns, the algorithm constructs a mathematical model that captures the essence of the data. (ii) Once the model has been optimized, it's time to assess its performance. The model is applied to a new set of data called the "test set," one it hasn't seen before. The predictions on this set are compared to the actual values, providing an early measure of its accuracy and generalizability.

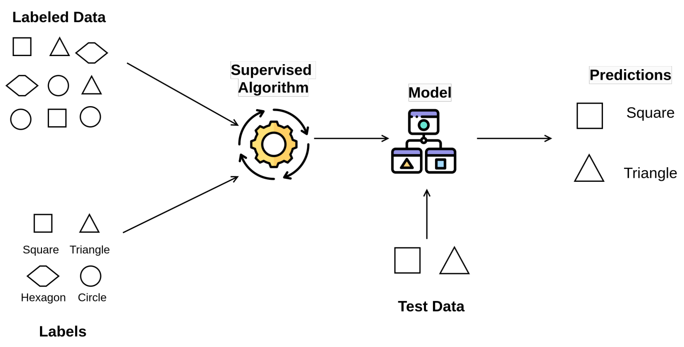


Figure 3.2: Training process of supervised learning algorithms.

3.2.1 Machine learning paradigms

ML encompasses various paradigms or approaches, each tailored to specific types of problems and data. This section discusses some key ML paradigms in detail.

Supervised learning. This type of learning is the most frequently used. In supervised learning, as shown in Figure 3.2, the algorithms are trained on labeled datasets to build a model. This data consists of pairs where each data point (input) is accompanied by a pre-defined label (output). The central objective of supervised learning is for the algorithm to learn the underlying patterns and relationships within the data, enabling it to make accurate predictions or classifications on new, unseen data. During the training phase, the algorithm iteratively adjusts its internal parameters to minimize the difference between its predictions and the actual labeled outputs. This process allows the model to generalize from the training data, making it capable of making informed decisions on novel inputs. During the testing phase, the model uses its learned map to assign labels to new data points. Supervised learning finds extensive applications in various domains, including image and speech recognition, natural language processing, medical diagnosis, and financial forecasting. The effectiveness of supervised learning lies in its ability to leverage labeled examples to build models that can generalize well to real-world scenarios, making it a cornerstone in ML research and applications. Artificial Neural Network (ANN), K-Nearest Neighbor (KNN), Decision Tree (DT) and Random Forest (RF) are some common supervised algorithms.

Unsupervised learning. Unsupervised learning takes place when a learning system is tasked with identifying patterns in data without the presence of labels (Figure 3.3). It involves training algorithms on unlabeled datasets, where the input data

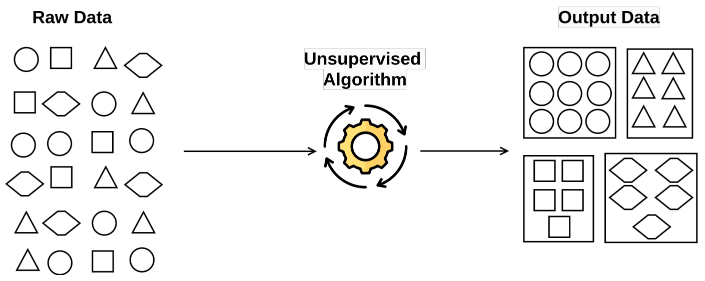


Figure 3.3: Training process of unsupervised learning algorithms.

lacks predefined output labels [37]. The primary goal of unsupervised learning is to uncover inherent patterns, structures, or relationships within the data without explicit guidance. Common techniques employed in unsupervised learning include: (i) Clustering, where the algorithm groups similar data points together into clusters, based on similarities or differences. Clustering techniques are used in a wide range of applications including fraud detection, market and customer segmentation, and recommendation engines. (ii) Dimensionality reduction which addresses the challenges posed by datasets with a high number of features or variables. It involves the process of reducing the number of input variables in a dataset while retaining the essential information. The primary goal of dimensionality reduction is to simplify the complexity of the data, making it more manageable for analysis and interpretation. It is particularly valuable for tasks such as visualization. (iii) Association rule mining, a technique in unsupervised learning that aims to discover interesting relationships or associations between variables in large datasets. It is often used in market basket analysis or recommendations. K-mean clustering, fuzzy clustering, and hierarchical clustering are some commonly used unsupervised algorithms.

Semi-supervised learning. Semi-supervised combines elements of both supervised and unsupervised learning. In this approach, the model is trained on a dataset that contains both labeled and unlabeled examples as illustrated in Figure 3.4. The labeled data provides explicit guidance for the model, allowing it to learn from known outcomes, while the unlabeled data presents an opportunity for the algorithm to discover underlying patterns or structures autonomously [38]. Semi-supervised learning is particularly useful in scenarios where obtaining labeled data is expensive or time-consuming. The model leverages the limited labeled examples to build a foundational understanding and then generalizes from the vast pool of unlabeled data. This methodology addresses the challenges of data scarcity and is applied in various domains, such as speech recognition, image classification, and natural lan-

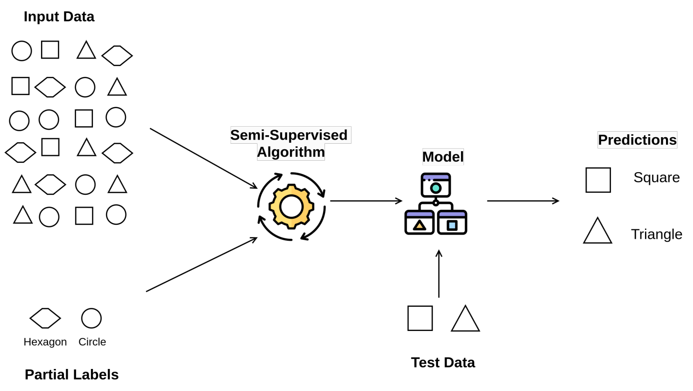


Figure 3.4: Training process of semi-supervised learning algorithms.

guage processing. Semi-supervised learning offers a practical compromise between the information-rich labeled data and the vast, often readily available, unlabeled datasets, providing a more efficient and cost-effective approach to training ML models. Graph-based methods [39], self-training [40], and generative models [41] are some of the most commonly employed techniques.

Reinforcement learning. Reinforcement learning is a dynamic and sophisticated paradigm in ML that revolves around the concept of an agent learning to make sequential decisions by interacting with an environment (Figure 3.5. In this framework, the agent receives feedback in the form of rewards or penalties based on the actions it takes. The primary objective of reinforcement learning is for the agent to learn optimal strategies that maximize cumulative rewards over time. The learning process involves exploring different actions, observing their outcomes, and adjusting the agent's decision-making policy to achieve better long-term outcomes. Key components of reinforcement learning include the environment, which represents the external system with which the agent interacts, the state, representing the current situation or configuration, and the action and reward functions, guiding the agent's behavior. Reinforcement learning has found application in various domains, including robotics, game playing, and autonomous systems like self-driving cars. Algorithms such as Q-learning [42] and deep reinforcement learning [43], where neural networks are employed, have contributed to significant advancements in this field, making it a crucial area of research for developing intelligent and adaptive systems.

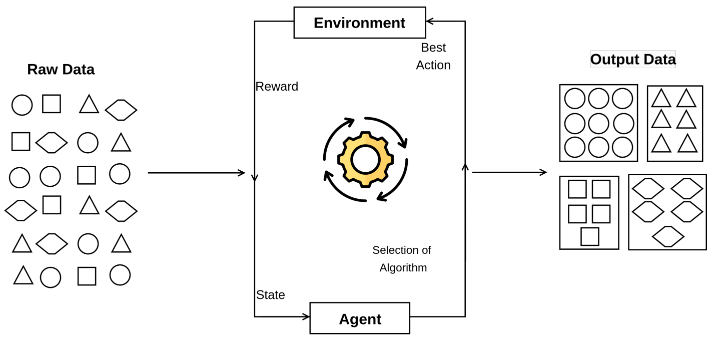


Figure 3.5: Training process of reinforcement learning algorithm.

3.2.2 Machine learning tasks

Classification. A classification task is a fundamental problem in supervised ML where the goal is to assign predefined categories or labels to input data based on its features. The input data, often referred to as instances, is characterized by a set of attributes or features, and the model is trained on a labeled dataset to learn the relationships between these features and the corresponding target labels. The trained model can then generalize its knowledge to make predictions on new, unseen instances. Classification tasks can be binary, where there are two possible classes (e.g., attack or normal), multi-class, involving more than two classes (e.g., identifying different types of attacks), or multi-labeled, where each instance can be assigned to multiple classes simultaneously (e.g. labeling a movie with multiple genres, such as action, comedy, drama, etc.). Common algorithms for classification tasks include logistic regression, decision trees, support vector machines, and neural networks. The performance of a classification model is typically assessed using metrics such as accuracy, precision, recall, and F1-score, which evaluate the model's ability to correctly classify instances across different classes. Classification tasks are pervasive in numerous real-world applications, including image and speech recognition, sentiment analysis, medical diagnosis, and fraud detection.

Regression. A regression task is also based on supervised ML that involves predicting a continuous numerical value or output based on input features. Unlike classification tasks that focus on assigning instances to predefined categories, regression aims to model the relationship between input variables and a continuous target variable. Regression tasks find widespread application in various domains, including finance for predicting stock prices, healthcare for estimating patient outcomes, and engineering for modeling physical phenomena. The versatility of regression makes it a vital component in predictive modeling and data analysis.

Clustering. Clustering falls under the umbrella of unsupervised learning. The primary objective of clustering is to group similar instances based on inherent patterns or similarities within the data, without prior knowledge of the class labels. In this task, the algorithm autonomously identifies structures in the dataset, forming clusters where data points within the same cluster share common characteristics. Clustering has diverse applications, ranging from customer segmentation in marketing to image segmentation in computer vision. It is a crucial tool for discovering hidden structures, gaining insights into the inherent organization of data, and facilitating subsequent analyses in different domains. The evaluation of clustering algorithms involves metrics such as silhouette score [44] and the Davies-Bouldin index [45], which assess the quality of the formed clusters.

3.2.3 Evaluation metrics

Once a ML model is trained, it must be evaluated to test its performance and effectiveness. Several evaluation metrics can be used to assess the model's performance. These metrics provide quantitative measures that help determine how well a model generalizes to unseen data or accomplishes specific tasks. The choice of metrics depends on the nature of the ML problem. In the following, we introduce the main metrics often used for classification and regression tasks.

Before introducing the metrics, we define the terms used to calculate them:

True Positive (TP): Instances that are correctly predicted as positive by the model.

False Negative (FN): Instances that are incorrectly predicted as negative by the model when they are positive.

True Negative (TN): Instances that are correctly predicted as negative by the model.

False Positive (FP): Instances that are incorrectly predicted as positive by the model when they are negative.

- Accuracy (ACC): Measures the ratio of correctly predicted instances to the total instances and is calculated as :

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

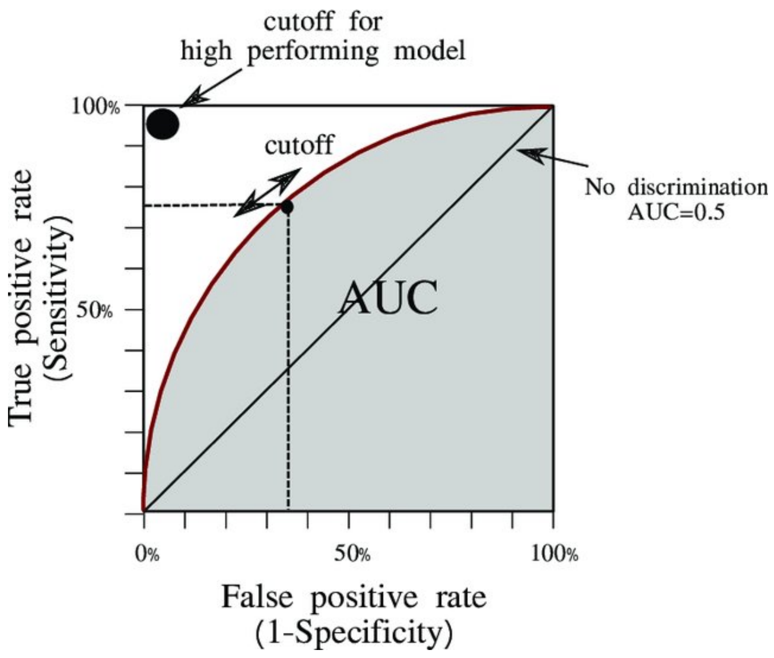


Figure 3.6: ROC curves and area under the curve (retrieved from Rodríguez-Hernández et al. 2021).

- Precision: Quantifies the accuracy of positive predictions and is defined as:

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

- Recall (Sensitivity): Measures the ability to capture all positive instances and is given by :

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

- F1-Score: The harmonic mean of precision and recall, providing a balance between the two, calculated as :

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

- Area under the ROC Curve (AUC-ROC): Represents the area under the Receiver Operating Characteristic curve, illustrating the trade-off between true positive rate and false positive rate (see Figure 3.6).

In Regression tasks, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are common:

- Mean Squared Error (MSE): a common metric used in regression tasks to measure the average squared difference between the predicted values and the actual values. It provides a way to quantify the overall accuracy of a regression model. The formula for calculating MSE is as follows:

  $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  Where n is the number of instances in the dataset. y_i represents the actual (ground truth) value of the target variable for the ith instance. $\hat{y}_i$ represents the predicted value of the target variable for the ith instance.

- Mean Absolute Error (MAE): It quantifies the average absolute difference between the predicted values and the actual values. It provides a straightforward measure of the average magnitude of errors, without emphasizing outliers as much as MSE. MAE is calculated as follows:

  $$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

3.2.4 Shallow models

Artificial Neural Network (ANN). ANNs, illustrated in Figure 3.7, are computational models inspired by the structure and function of the human brain, designed to simulate the way biological neural networks process information. ANNs consist of interconnected nodes, or "neurons", organized in layers: an input layer, one or more hidden layers, and an output layer. Each connection between nodes has an associated weight, and each node applies an activation function to the weighted sum of its inputs. Through a process known as forward propagation, input data is passed through the network to produce an output. Training an ANN involves adjusting the weights during a process called backpropagation, where the model learns from the difference between predicted and actual outputs. ANNs can be used for both regression and classification tasks and have demonstrated remarkable success in various domains, such as image and speech recognition, natural language processing, and autonomous systems.

Convolutional Neural Network (CNN). CNNs are a type of neural network designed for processing and analyzing visual data (Figure 3.8). CNNs are inspired by

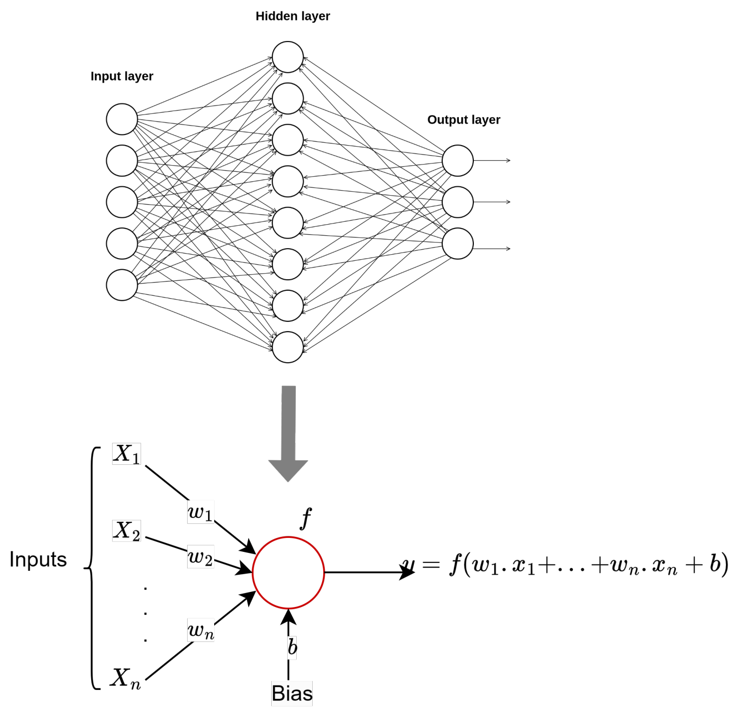


Figure 3.7: Structure of a shallow ANN.

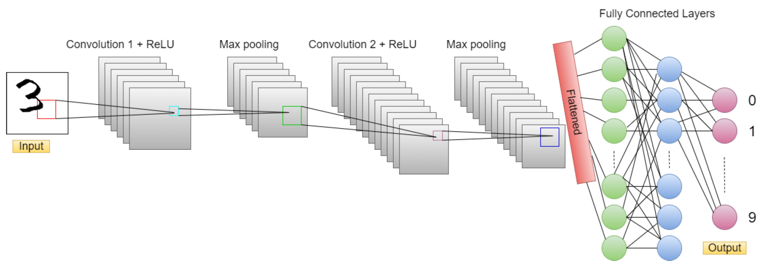


Figure 3.8: Structure of a CNN network.

the human visual system, utilizing convolutional layers to automatically learn hierarchical representations of features from input images. The key components of a CNN include convolutional layers, pooling layers, and fully connected layers. In the convolutional layers, filters are applied to the input data to detect patterns and spatial hierarchies. Pooling layers reduce the spatial dimensions, capturing the most essential information. Fully connected layers connect all neurons from one layer to another, enabling high-level feature extraction and classification. CNNs are particularly effective in tasks related to image recognition, object detection, and classification and have demonstrated remarkable success in various computer vision tasks, such as autonomous vehicles, and medical image analysis.

Decision Tree (DT). DTs are a type of shallow ML models used for both classification and regression tasks. These models represent a tree-like structure where each internal node represents a decision based on a specific feature, each branch represents the outcome of that decision, and each leaf node represents the final prediction or outcome. The decision-making process involves recursively partitioning the dataset based on the most informative features to maximize the homogeneity of data within each resulting subset.

In classification tasks, DTs are designed to classify instances into predefined categories, while in regression tasks, they predict a continuous numerical value. Unlike black-box models, DTs are readily interpretable. You can follow the branches, understand the logic behind each split, and gain valuable insights into the relationships between features and outcomes. Moreover, they require little data preprocessing and can handle both numerical and categorical features as well as noisy data. Therefore, their performance is less susceptible to outliers than some complex models. But like any other model, DTs have their limitations. They are susceptible to overfitting, especially with deep trees, and may not generalize well to unseen data.

To mitigate overfitting, techniques such as pruning and setting a minimum number of samples per leaf node can be employed. Ensemble methods like Random Forests, which use multiple DT, address some limitations of individual DT, providing improved performance and robustness. DTs remain a fundamental building block in ML, forming the basis for more complex models and contributing to the interpretability of the overall model.

Random Forest (RF). RF [46] is an ensemble learning method that belongs to the tree family. They are composed of a collection of DTs, where each tree is trained on a different subset of the data. Each tree uses a random selection of features at each split, which helps reduce the correlation between trees and increase diversity in the ensemble. During classification tasks, each tree in the random forest predicts a class label, and the final decision is made based on a majority vote of all the individual trees. For regression tasks, the final prediction is the average of the predictions from all trees. This ensemble approach reduces variance, prevents overfitting, and leads to more accurate and stable predictions compared to a single decision tree. Overall, random forests are known for their robustness and strong performance across various domains.

Extra Tree (ET). ET [47], short for Extremely Randomized Trees, is also an ensemble ML algorithm that extends the concept of RF. Like RF, ET builds multiple DTs during training to improve predictive performance and robustness. The key distinction lies in how they create data subsets and introduce randomness during the tree-building process. In ET, the splitting of nodes is performed randomly, using not only random subsets of features but also random threshold values for splitting nodes. This extra layer of randomness promotes diversity among the trees and can enhance the model's generalization capabilities.

Let $T_1, T_2, \ldots, T_N$ represent the individual DT, and given a new input X_{new}, the ET's classification decision y_{final} is determined by:

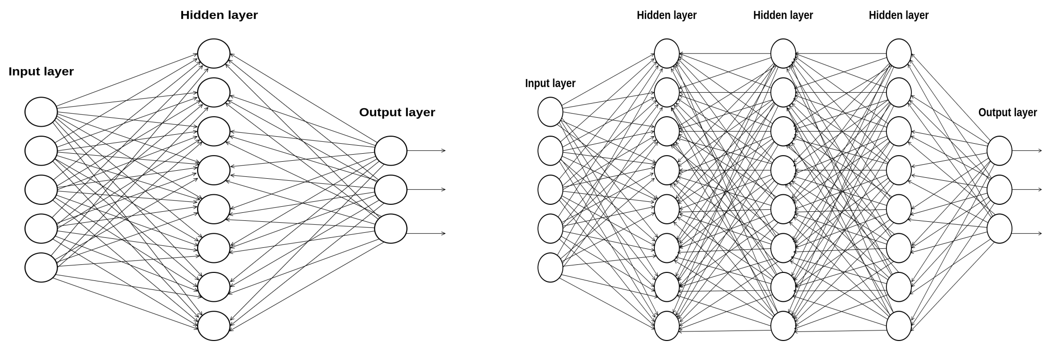
$$y_{final} = mode(T_1(X_{new}), T_2(X_{new}), \ldots, T_N(X_{new})) \tag{3.1}$$

ET is often used in classification tasks and is computationally efficient, making it suitable for large-scale datasets. The same representation of the RF model holds for the ET model as both are ensemble methods based on DT.

3.3 Deep learning

DL is a subset of the broader field of ML, characterized by a neural network with two or more hidden layers (*hidden layers* = L and L $\geqslant$ 2), as shown in Figure 3.9. These networks aim to imitate the functions of the human brain, enabling them to learn from extensive datasets. The distinguishing aspect of DL lies in its capacity to independently learn complex patterns and representations from large datasets. Unlike traditional ML approaches, DL models excel at extracting features through multiple layers, allowing them to reveal hierarchies of information within raw data. This ability has proven particularly valuable in tasks such as image and speech recognition and natural language processing [48, 49], where identifying complex patterns is crucial. Moreover, unsupervised training is another key feature of DL. The latter aims to learn from extensive sets of unlabeled data, enabling the training of models in a fully unsupervised manner. Subsequently, if labeled data is available, often limited in quantity, then it is employed to fine-tune the model in a supervised classification setting.

DL techniques can be categorized into three primary architectures, generative architectures (unsupervised), discriminative architectures (supervised), and hybrid. The primary goal of generative architectures is to model the underlying distribution



(a) Non-deep Neural Network (b) Deep Neural Network

Figure 3.9: The difference between non-deep and deep neural networks.

of the input data. These models aim to learn the inherent structure and patterns within the dataset without explicit labels. They generate new data samples that resemble the training data, capturing the intrinsic characteristics of the dataset. Common examples include Genetative Adversarial Networks (GANs) and AEs. Generative architectures find applications in image synthesis, data augmentation, and generating realistic samples in domains like computer vision and natural language processing. Discriminative architectures focus on learning the boundary between different classes or categories within the labeled dataset. The emphasis is on distinguishing patterns associated with specific labels. These models are adept at classification tasks, making predictions on the class or label of a given input. Common examples include CNNs for image classification and Recurrent Neural Networks (RNNs) for sequence labeling. Discriminative architectures are widely used in tasks such as image recognition, speech recognition, and natural language processing where the goal is to classify inputs into distinct categories. Hybrid architectures integrate both generative and discriminative components, leveraging the strengths of each. This combination aims to enhance overall model performance and versatility. By incorporating generative and discriminative elements, hybrid architectures can handle a broader range of tasks. For instance, they may generate new samples while also being proficient in classification tasks. Hybrid architectures are applied in scenarios requiring a balance between data generation and classification, offering flexibility in tasks that involve both unsupervised and supervised aspects. Figure 3.10 summarizes the DL techniques for each category.

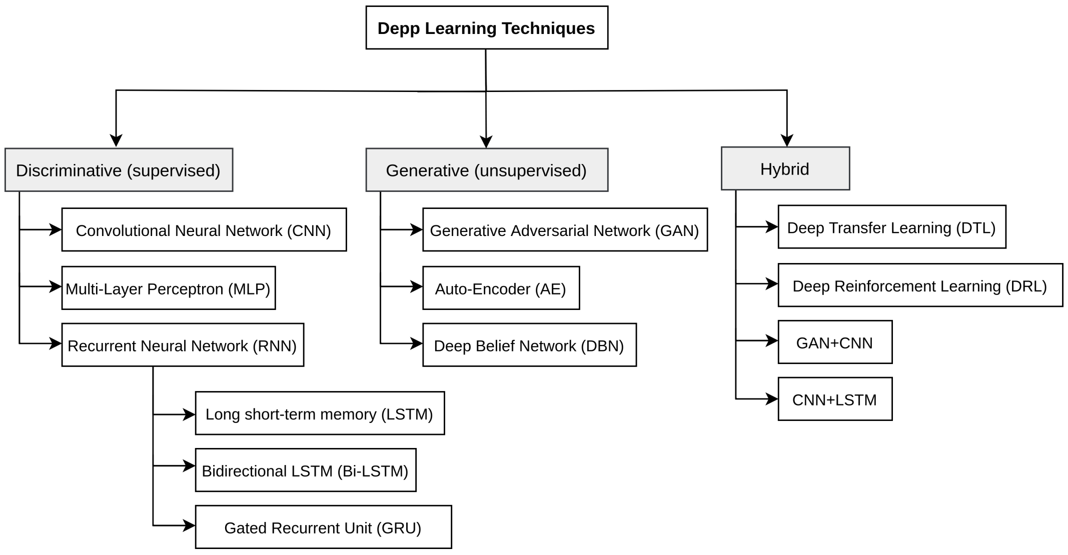

Depp Learning Techniques
Discriminative (supervised)
Generative (unsupervised)
Hybrid
Convolutional Neural Network (CNN)
Multi-Layer Perceptron (MLP)
Recurrent Neural Network (RNN)
Long short-term memory (LSTM)
Bidirectional LSTM (Bi-LSTM)
Gated Recurrent Unit (GRU)
Generative Adversarial Network (GAN)
Auto-Encoder (AE)
Deep Belief Network (DBN)
Deep Transfer Learning (DTL)
Deep Reinforcement Learning (DRL)
GAN+CNN
CNN+LSTM

Figure 3.10: Taxonomy of deep learning techniques.

3.3.1 Deep learning models

Multi-Layer Perceptron (MLP). MLP [50] is a type of feedforward ANN with three or more hidden layers and is widely employed in various domains due to its capability to model complex non-linear relationships between input features and target classes. Given input features $X = [x_1, x_2, \dots, x_n]$, where n is the number of input features, and assuming a single hidden layer h with p neurons and an output layer with m neurons (for classification tasks with m classes) where Y represents a vector of output labels. The MLP classifier can be expressed as follows:

Hidden layer output:

$$h = \sigma(W_1 X + b_1) \tag{3.2}$$

Output layer output:

$$Y = \text{softmax}(W_2 h + b_2) \tag{3.3}$$

where σ represents the chosen activation function for the hidden layer, such as the sigmoid function, hyperbolic tangent function, or ReLU [51, 52], W_1 and W_2 are weight matrices of size $p \times n$ and $m \times p$ respectively, for the connections between layers, and b_1 and b_2 are bias vectors for the respective layers of size p and m respectively.

Autoencoder (AE). An autoencoder [53] is a specific type of unsupervised neural network that learns an "informative" representation of input data. Let $x \in \mathbb{R}^d$ be the input and $z \in \mathbb{R}^p$ the latent representation of x, such that $p \ll d$. Simple AEs [54] encompass two blocks, an encoder, and a decoder. The former is a neural network that learns a latent representation z of x. The latter is also a neural network that uses the vector z generated by the encoder to reconstruct the input data with minimal loss error, such that $q_\theta(z|x)$ and $p_\phi(x|z)$ are the encoding and decoding distributions, respectively. Variational Autoencoders (VAEs) [55] are similar to simple AEs, nevertheless, they map the input vector to a distribution of mean μ and standard deviation σ from which the latent vector z is sampled. Figure 3.11 illustrates the general architecture of simple AEs and VAEs.

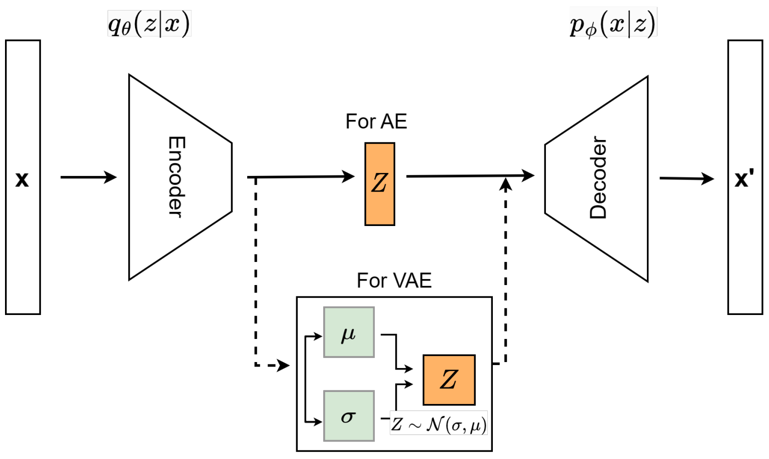


Figure 3.11: The general architecture of simple autoencoders and variational autoencoders.

Adversarial Autoencoders (AAEs) [56] are hybrid models that fuse AEs and GANs [57]. It uses the adversarial loss concept introduced by GANs to improve the regularization of an autoencoder. More specifically, an AAE employs adversarial learning in order to impose the latent space to follow a certain distribution that could be a p-dimensional normal distribution $\mathcal{N}(0, I)$. The overview of AAEs architecture is illustrated in Figure 3.12, where $p(z)$ is the prior distribution we want to impose on z, and $q(z)$ is the distribution of the latent variable. The top half of Figure 3.12 is a standard autoencoder that plays the role of the generator and z is the generated data. The generator attempts to fool the discriminator into believing that the latent representation is sampled from the pre-chosen distribution. On the other hand, the discriminator $D_\chi(z)$, depicted in the bottom half of the figure, predicts whether a given latent vector is generated by the encoder (fake) or sampled from the predefined distribution (real). Both the autoencoder and the adversarial network are trained jointly with Stochastic Gradient Descent (SGD) in two phases: the *reconstruction phase* and the *regularization phase*. In the reconstruction phase, only the parameters of the encoder and decoder are optimized in order to minimize the reconstruction loss of the inputs, while in the regularization phase, both the discriminator and the generator (encoder) are trained at once. First, the discriminator learns how to distinguish between real samples (drawn from the prior) and fake samples (generated by the encoder). Then, the discriminator is fixed and the generator is trained to acquire the ability to produce samples following the prior.

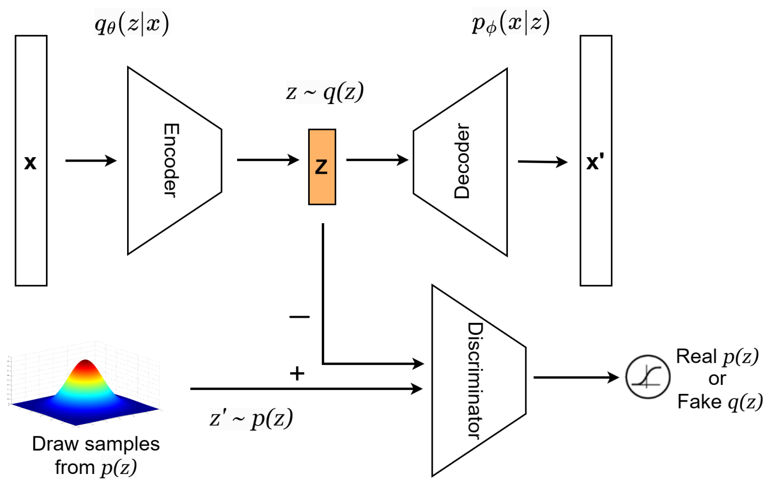


Figure 3.12: The general architecture of adversarial autoencoders.

Generative Adversarial Network (GAN). GANs are a type of generative model, first introduced by Ian Goodfellow and his colleagues in 2014 [57]. As depicted in Figure 3.13, a GAN consists of two deep neural networks, a generator, and a discriminator, competing against each other in a game-theoretic scenario. The generative network (G) tries to generate realistic samples to deceive the discriminative network (D), while D tries to accurately distinguish between the real and generated samples. The ultimate goal is for the generative network G to generate samples that are indistinguishable from real data, while the discriminative network D aims to correctly classify the samples as real or generated. Therefore, both the generator and the discriminator engage in a game referred to as a "two-player minimax game", that is, G must be minimized and D must be maximized in the following function:

$$\begin{gathered} \min_G \max_D V(D,G) \\ V(D,G) = E_{x \sim p_{data}(x)}[log D(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))] \end{gathered} \tag{3.4}$$

$G(z)$ is the synthetic data produced by the generator and $D(x)$ is the probability that the data x is real or fake predicted by the discriminator. The generator attempts

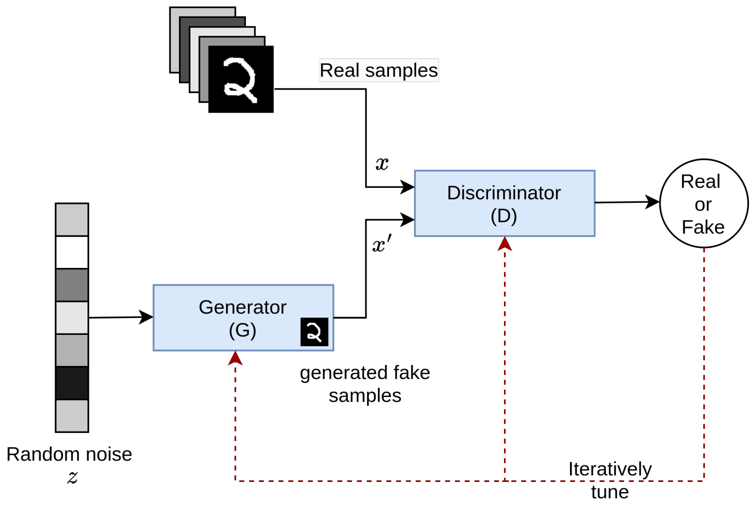


Figure 3.13: The structure of the GAN network.

to minimize the probability of misclassification by the discriminator, and the discriminator seeks to maximize its ability to distinguish between real and generated samples. This adversarial training dynamic results in the refinement of both networks.

Bidirectional GAN (BiGAN). Following the introduction of GANs, numerous variations and extensions have been proposed to enhance and adapt the original framework to different tasks and challenges. One notable variant is the Bidirectional GAN (BiGAN), which introduces a two-way learning strategy by extending the GAN with an encoder network (E) alongside the generator and discriminator. BiGAN introduces an additional term to the GAN objective:

$$\underset{G,E}{min}\ \underset{D}{max}\ E_{x\sim p_{data}(x)}[logD(x,E(x))] + E_{z\sim p_z(z)}[log(1-D(G(z),z))] \tag{3.5}$$

Here, $E(x)$ aims to map real and generated samples back to a shared latent space, promoting a bidirectional relationship between the data space and the latent space. This addition facilitates a more comprehensive understanding of the latent space and enables applications such as image-to-image translation and anomaly detection.

3.4 Federated learning

FL was first introduced in 2016, by McMahan et al. [58], as a decentralized ML framework where multiple entities jointly train a global statistical model $f(w)$ without the need to send the local data to a centralized server. At each communication round of

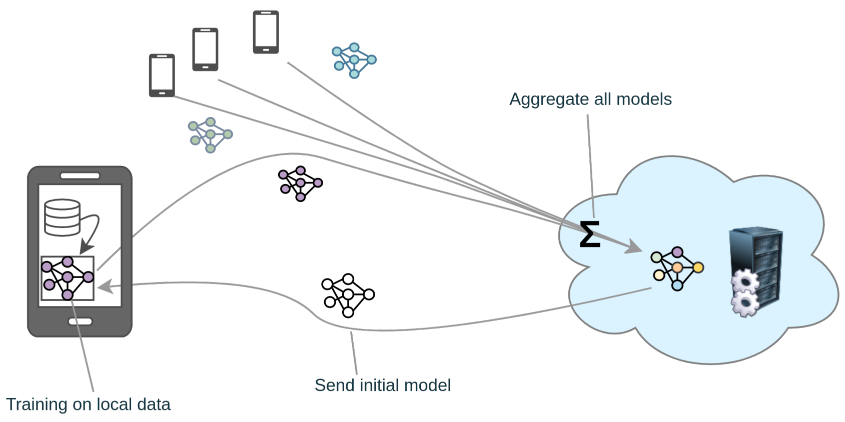


Figure 3.14: General architecture of federated learning.

the training, a fraction of the clients are chosen to collaboratively minimize an objective function $F(w)$:

$$\min_{w} F(w) = \min_{w} \sum_{k=1}^{K} p_k F_k(w) \tag{3.6}$$

where $F_k(w) = \frac{1}{n_k} \sum_{j_k=1}^{n_k} f_{j_k}(w; x_{j_k}, y_{j_k})$ is the local objective function measuring the empirical risk over local data, K is the number of entities, n_k is the number of data points available at the entity k, and $p_k = \frac{n_k}{n}$ is the relative impact of each entity such that $n = \sum_k n_k$ is the total number of data samples.

More specifically, in the iterative process of FL, each round involves the strategic selection of a subset S_K from the total K client devices, each holding its own local dataset. Within this subset, each client k downloads the current global model w_t from the central server and optimizes it locally on its unique dataset D_k, resulting in an updated model w_{t+1}^k. Post local training, only the model updates, reflecting the learned knowledge, are communicated back to the central server. The central server aggregates these updates to construct an improved global model, capturing insights from diverse local datasets. This iterative and collaborative process is repeated across multiple rounds, enabling the global model to progressively refine itself without necessitating the central storage or exchange of raw data. As illustrated in Figure 3.14 FL encompasses the following phases:

1. **Initialization:** In this phase, the server selects the set of clients to participate in the current round. In addition, the server initializes the global model either with pre-trained weights or random values, which is then distributed to participating clients.

2. **Local Training:** Each client updates the model locally on its own dataset. This local training involves updating the model parameters using optimization algorithms like SGD. The model learns from the specific patterns and features present in the local data.

3. **Model Update:** After local training, each device computes the difference between its locally trained model and the global model. This difference is often represented as a set of gradients or parameter updates.

4. **Aggregation:** The model updates from all devices are sent to the central server (aggregator). The latter aggregates these updates to update the global model. Common aggregation methods include averaging or weighted averaging.

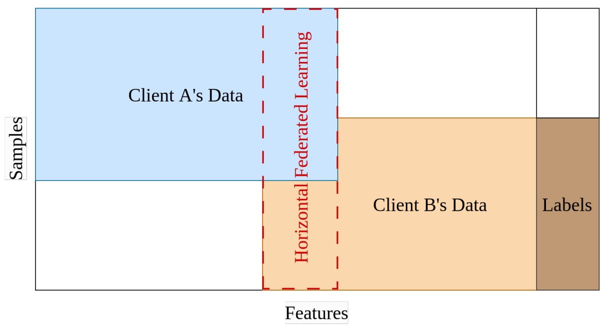
5. **Iteration:** Finally, steps 2-4 are repeated for multiple rounds or until a convergence criterion is met. Each iteration allows the global model to be refined based on the collective knowledge of all devices.

The key defining characteristic of FL is its adept handling of privacy concerns; raw data remains securely on local devices, with only aggregated model updates shared. This privacy-preserving and collaborative framework proves particularly valuable in scenarios characterized by sensitive, distributed, and heterogeneous data.

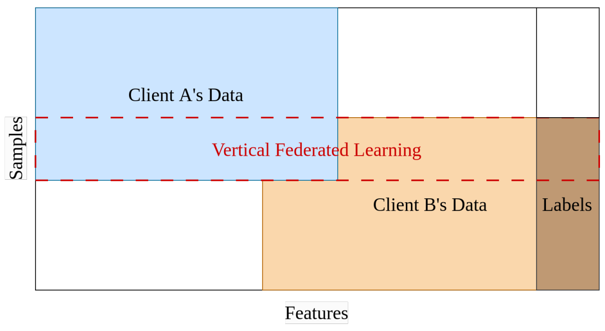
3.4.1 Categorization of federated learning

FL comes in various forms, according to how the clients' data are distributed. It can be categorized into three main types: Horizontal Federated Learning (HFL), Vertical Federated Learning (VFL), and Federated Transfer Learning (FTL) [59]. In this section, we discuss each type thoroughly.

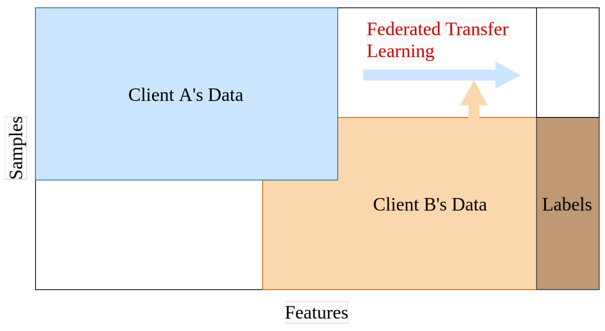
Horizontal Federated Learning (HFL). In HFL or sample-based FL (Figure 3.15(a)), each participating device or client holds data instances that share the same feature space but represent different entities. For instance, multiple hospitals may each possess patient data with similar attributes (e.g., age, blood pressure), but for different individuals. Clients collaboratively train a shared model using their locally stored data without sharing individual data points. Model updates are aggregated at a central server, ensuring that insights from diverse instances of the same features contribute to the global model's improvement. HFL excels in scenarios with large-scale



(a) Horizontal Federated Learning



(b) Vertical Federated Learning



(c) Federated Transfer Learning

Figure 3.15: Categorization of federated learning.

datasets, such as personalized smartphone applications and image recognition systems. However, HFL may not be suitable for tasks requiring domain-specific knowledge or when data across devices exhibits significant heterogeneity.

Vertical Federated Learning (VFL). VFL or feature-based FL involves clients with complementary information (Figure 3.15(b)). In this setting, data instances share some common features but differ in others. For example, one client may have demographic data, while another has medical records, and both are necessary for a comprehensive analysis. Instead of sharing entire data instances, clients share specific features while keeping others private. This collaborative model training allows the global model to learn from a merged dataset without exposing complete information. Secure techniques like homomorphic encryption may be employed to facilitate feature sharing without revealing the raw data. VFL is particularly beneficial for healthcare applications, where hospitals can collectively train a disease prediction model without sharing sensitive patient data.

Federated Transfer Learning (FTL). FTL combines HFL and VFL in scenarios where both the sample and feature spaces are different (Figure 3.15(c)). FTL focuses on leveraging knowledge gained from one task or domain to improve learning on another related task or domain. It involves transferring insights from one set of clients to benefit the training of a model on a different set of clients. A model pre-trained on a source task or domain is transferred to a target set of clients. The model is then fine-tuned or adapted to the specific characteristics of the new data distribution. This helps in situations where labeled data is scarce in the target domain but abundant in a related source domain. FTL offers promising applications in mobile app recommendation systems, where pre-trained models can be fine-tuned based on individual user data to provide personalized recommendations. However, FTL's effectiveness depends heavily on the availability of a suitable pre-trained model and may not be as effective when data deviates significantly from the pre-training data.

Each form of FL addresses specific challenges associated with different data distribution patterns, ensuring that models can be trained collaboratively without compromising privacy or data security. The choice between horizontal, vertical, or transfer FL depends on the nature of the data and the objectives of the collaborative learning task.

3.4.2 Federated learning algorithms

Since its introduction in 2016, the field of FL has witnessed a prolific surge in algorithmic innovation. FL has seen a dynamic and expansive landscape of research, resulting in the proposal of a myriad of FL algorithms. These algorithms address diverse challenges ranging from communication efficiency and scalability to the accommodation of non-IID data (cf. Section 3.4.3). The rest of this section discusses some of the current state-of-the-art FL algorithms.

FedAvg. *FedAvg* the first and the standard FL algorithm proposed by the researchers who introduced FL [58]. *FedAvg* is a simple yet effective algorithm designed to compute a global model based on the weighted average of the parameters collected from clients. The model is then sent back to the clients for further training. At its core, *FedAvg* involves an iterative process wherein individual devices locally train on their datasets using a shared global model. The model updates are computed over a batch $b \in \mathcal{B}$ as:

$$w \leftarrow w - \eta \nabla \mathcal{L}(w; b) \tag{3.7}$$

where $\mathcal{L}$ represents the local loss function and η is the learning rate. The global model is updated by averaging these local updates:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^{k} \tag{3.8}$$

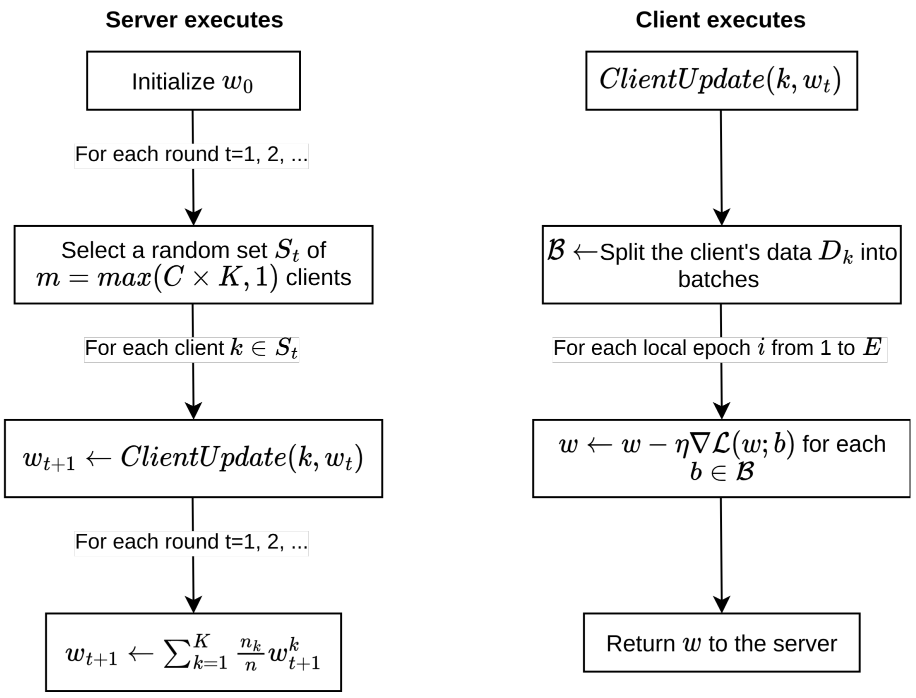
Figure 3.16 provides a visual summary of the *FedAvg* algorithm, illustrating the key steps and interactions between the client and server in a FL setting.

Fedavg is easy to implement and scales well to large datasets and FL networks with a large number of clients. However, *Fedavg* faces challenges when the data across clients is non-IID [60].

FedProx. *FedProx* algorithm (Federated Proximal Gradient Descent) [61] is a generalized version of *FedAvg* that tackles the statistical heterogeneity issue within the FL environment. *FedProx* modifies *FedAvg* by adding a regularization term δ to the local objective functions to limit the impact of the local updates and thus restrain the local models' divergence, especially in non-IID scenarios. The regularization term $\frac{\delta}{2}\|w - w_t\|^2$ is added to the loss function of each client, where w are the parameters of the global model, w_t are the parameters of a local model at round t, and $\delta \in [0, 1]$ is a regularizer factor. Hence the new objective function to minimize is $F'(w)$:

$$\underset{w}{min}\, F'(w) = F_k(w) + \frac{\delta}{2} \|w - w_t\|^2 \tag{3.9}$$

FedProx reduces divergence and achieves faster convergence compared to *FedAvg*, especially in non-IID settings. Furthermore, the server-side aggregation and regularization further stabilize learning and lead to better accuracy on non-IID datasets.



Server executes
Initialize w_0
For each round t=1, 2, ...
Select a random set S_t of $m = max(C \times K, 1)$ clients
For each client $k \in S_t$
$w_{t+1} \leftarrow ClientUpdate(k, w_t)$
For each round t=1, 2, ...
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^{k}$
Client executes
$ClientUpdate(k, w_t)$
$\mathcal{B} \leftarrow$Split the client's data D_k into batches
For each local epoch i from 1 to E
$w \leftarrow w - \eta \nabla \mathcal{L}(w; b)$ for each $b \in \mathcal{B}$
Return w to the server

Figure 3.16: The standard federated learning algorithm *FedAvg*.

However, choosing the optimal penalty term for proximal updates requires careful tuning based on the dataset and model complexity.

FedMA. *FedMA* (Federated Matched Averaging) [62] proposes a layer-wise aggregation approach that tackles the non-IID challenge through three key steps: 1) Matching: during each round, similar neurons across local models are identified based on their feature extraction signatures. This matching process accounts for data variations across devices. 2) Averaging: the matched neurons are then fused via weighted averaging, preserving information from individual devices while reducing redundant updates. 3) Adaptive growth: in addition to averaging, *FedMA* can discover and add entirely new neurons to the global model based on the collective knowledge of the devices. This facilitates capturing features unique to the aggregated data, further enhancing model performance. By considering the specific features of each device, *FedMA* can achieve better accuracy when data is not identically distributed. Moreover, by merging similar neurons and potentially adding new ones, the amount of data transmitted between devices can be reduced. However, with *FedMA*, the Complexity increases as matching and adaptive growth involve more sophisticated computations compared to *FedAvg*, potentially posing challenges for resource-constrained devices.

FedNova. *FedNova* [63] tackles the non-IID challenge through a weighted aggregation approach. Each device trains a local model on its own dataset. Then, each local model update is normalized by the number of training samples used on that device, adjusting for data imbalances. Finally, the normalized updates are then aggregated at the server using a weighted average, where devices with more training data contribute more to the global model update. By correcting for data imbalances, *FedNova* reduces bias and achieves better accuracy on non-IID datasets compared to *FedAvg*. In certain scenarios, *FedNova* can converge to optimal performance faster than *FedAvg* due to the weighted averaging mechanism. Additionally, compared to techniques requiring additional communication for personalization or server-side aggregation, *FedNova* maintains the same communication cost as *FedAvg*. Nevertheless, normalizing updates requires additional computation compared to *FedAvg*, potentially impacting resource-constrained devices.

FedBN. *FedBN* [64] addresses the issue of Batch Normalization (BN) in FL. The authors proposed *FedBN* that keeps all BN parameters at the clients and updates them locally without communication with the server. The empirical results presented in the paper have shown that *FedBN* performs better than standard *FedAvg*. However, the paper does not study the efficiency of *FedBN* in the case of unbalanced labels and for more complex datasets, moreover, the paper does not cover the case where the testing clients have no data to compute BN statistics on.

3.4.3 Federated learning challenges

FL is regarded as a seminal work and was adopted by many applications such as Google Home and Amazon Alexa. However, FL has induced three main challenges:

1. Communication bottleneck [65, 66, 67]: Although the computation power of mobile devices has increased significantly in the last decade, the bandwidth of wireless communications has not improved greatly. The bottleneck has then shifted from computation into communication in which the limited computation bandwidth has significantly slowed down the convergence time.

2. Systems heterogeneity [68]: The data-owners participating in FL process generally vary in terms of system-level characteristics (i.e. storage, computational power, communication bandwidth, etc) which raises issues such as straggler mitigation and fault tolerance.

3. Statistical heterogeneity [69, 70]: Intuitively, each device in the FL setting, has local data samples that are biased by the unique device user environment and characteristics. Therefore, in practice, local data on edge devices is not always IID. As a result, the IID assumptions are violated in FL and common architectures tend to perform poorly in this case. In the next section 3.5 we further discuss the non-IID data distribution.

3.5 Learning on non-IID data

In probability theory and statistics, IID refers to the assumption that each data point in a dataset is independent of other data points and that all data points are sampled from the same distribution. A scenario where the "independence" condition is not met is when the generation of the next sample is dependent on the preceding samples. In financial markets, for instance, stock prices are often influenced by past prices and market conditions. The next-day stock price is not independent of the previous day's price; it is influenced by factors such as historical trends, trading volumes, and macroeconomic indicators. The violation of "identicalness" occurs when data points are not sampled from the same distribution. For instance, the testing data comes from an entirely different distribution from the training data.

Let us consider a learning task $\mathcal{T}$ with features x and labels y. $\mathcal{P}_i$ is the data distribution of the client i. In non-IID settings, the difference between two distributions $\mathcal{P}_i$ and $\mathcal{P}_j$ for different clients i and j can be classified into various classes [71]:

Violation of Identicalness

- *Feature distribution skew:* The marginal distributions $\mathcal{P}_i(x)$ may vary across clients. The input features are not evenly distributed between clients. For instance, in a handwriting recognition task, users write the same words differently.

- *Same label, different features (concept shift):* The conditional distributions $\mathcal{P}_i(x|y)$ may vary across clients. Different features across different clients might be labeled with the same label. This is due to cultural differences and standards of living, etc. For example, in some regions, "Panda" is called "Panda" and in other regions, "Red Panda" is also called "Panda".

- *Same features, different labels (concept shift):* The conditional distributions $\mathcal{P}_i(y|x)$ may vary across clients. In this case, the same features across different clients are labeled differently. For instance, "Leopards" can either be called Leopards or "Pards".

- *Unbalancedness:* Different clients can hold vastly different amounts of data.

Violation of Independence

- *Inter-partition correlation:* Existing data on a client might be dependent on other existing data on another client. For example, for a phenomenon to be fully understood, all the information about the different devices is needed.
- *Intra-partition correlation:* Data samples held by a single partition are dependent. For example, the consecutive frames in a video are highly correlated.

The IID assumption is commonly made in statistical modeling and ML algorithms, as it simplifies the analysis and transforms complex real-world problems into tractable and quantifiable tasks, thus allowing for certain statistical techniques to be applied. Nevertheless, this simplification may also induce major limitations including biased solutions, lack of generalization, and over-simplification. The assumption of IID reflects a limited comprehension of the characteristics of real-world systems, not fully capturing systems' complexities. Consequently, solutions derived under IID may be biased and work only in very specific situations, and fail to generalize when the IID property does not hold. The latter is true in most real-world scenarios, where datasets may exhibit complex correlations or dependencies, imbalanced distributions, and/or temporal correlations between consecutive observations or neighboring data points. More specifically, ML models trained under IID can be biased towards the training data, leading to inaccurate predictions when deployed in real-world situations with differing data characteristics. Furthermore, they often struggle to generalize to unseen data or situations outside the narrow assumptions of the training data. This makes them less adaptable and unreliable in complex, real-world applications.

One scenario where non-IID data is common is FL. Intuitively, each client in the FL setting has local data samples that are biased by the unique client's environment and characteristics. Therefore, in practice, local data on edge devices is not always IID. As a result, the IID assumptions are violated in FL and common architectures tend to perform poorly in this case. The presence of non-IID data poses significant challenges to FL. First, the global model struggles to find a common denominator across diverse data. Each local dataset contributes unique insights into the underlying patterns, creating a scenario where there is no one-size-fits-all solution for the entire dataset. This struggle results in suboptimal performance, as the model might not capture the intricacies of any specific data distribution accurately. A second challenge is related to gradient divergence. The gradients computed during local model updates can diverge significantly. This divergence arises from the varied nature of data distributions, causing each device to provide gradients that point in different directions. The consequence is a lack of consensus in the learned model, hindering the convergence process and making it challenging to reach a stable global model. The challenge of gradient divergence is exacerbated when certain devices have unique or outlier patterns in their data, introducing conflicting gradients. Lastly, communication overhead becomes a concern as achieving convergence may require more rounds of communication due to the varied data representations. The frequent communication between devices and the central server for model updates becomes inefficient and resource-intensive, particularly in large-scale deployments with non-IID data. This communication overhead, driven by the diverse nature of data distributions among devices, amplifies latency and bandwidth consumption, posing scalability and efficiency challenges in FL implementations.

Although McMahan et al. [72] have shown the robustness of *FedAvg* towards certain non-IID distributions, several studies have demonstrated the divergence and instability of *FedAvg* in such setting. For instance, the experiments presented in [73] have shown a significant performance degradation coupled with a communication cost of *FedAvg* with a highly skewed non-IID data where the accuracy of neural networks was reduced up to 11% for MNIST, 51% for CIFAR-10 and 55% for keyword spotting (KWS) datasets. Similarly, Li et al. [60] have analyzed the convergence of *FedAvg* in a more realistic setting where the data is non-IID and the device participation rate is low in each round. Both theoretical studies and empirical results have shown that the convergence rate is gravely hindered in such circumstances.

To address this challenge, researchers have explored various strategies and optimizations aimed at mitigating the non-IIDness issue in FL under some assumptions. Some approaches involve the use of additional data, either held by the server or synthetic data generated by clients. For instance, Zhao et al. [73] proposed an enhanced version of *FedAvg* based on a data-sharing approach. The server holds a global dataset G drawn from a uniform distribution and shares a random small portion of G with all clients. Additionally, instead of starting with a global model initialized randomly, the server warms up the model centrally on the server-side proxy data. However, this strategy requires some public dataset being available for a particular task which might be unrealistic in practice. A similar approach [74] suggested the use of Federated Augmentations where each device can generate the missing samples using a generative model. However, this requires sharing some devices' data samples with the server, which violates the key privacy goal of FL. *FedProx*, *FedMa*, and *FedNova* algorithms that we discussed in 3.4.2 have also been proposed to enhance FL in non-IID settings.

In another line of work, [75, 76, 77, 78] allow a partial client participation based on a conditional client selection. In particular, [75] studies the convergence of *FedAvg* under biased client selection and proposes non-uniform clients' sampling based on their losses. The proposed strategy shows faster convergence and higher testing accuracy. However, such approaches are often not governed by the server but by the clients' availability.

3.6 Summary

In this chapter, we covered the field of ML. We discussed the different paradigms of ML, including supervised, semi-supervised, unsupervised, and reinforcement learning, as well as the three main tasks in ML, namely, classification, regression, and clustering. Then, we presented an overview of several shallow ML models. Additionally, we delved into the foundational concepts of DL, presented various DL models, and highlighted its superiority in learning features compared to shallow ML models. We further discussed the decentralized ML framework, FL, its categorizations, algorithms, and challenges. Finally, we shed light on one of the key challenges in ML, both in general and within the context of FL, and we reviewed previous works that tackled this issue. This chapter provided insights about the potential of ML, DL, and FL for resolving complex network challenges, specifically for intrusion detection, which is the aim of this thesis.

The subsequent chapter will discuss the deployment of ML/DL and FL for intrusion detection and will review existing research within this context.

4 ML-enabled Intrusion Detection

4.1 Introduction

The revolution of Machine Learning (ML) represents a transformative era in the realm of technology and data analytics. ML has revolutionized traditional approaches to problem-solving by enabling systems to learn from data, identify patterns, and make predictions or decisions without explicit programming. This paradigm shift has had a profound impact across diverse industries, unlocking new possibilities and efficiencies. In fields such as healthcare, finance, and autonomous systems, ML algorithms have demonstrated their effectiveness in extracting meaningful insights, optimizing processes, and enhancing decision-making. The revolution is characterized by the unprecedented scalability of ML models, allowing them to handle massive datasets and complex tasks with remarkable accuracy. The continuous evolution of ML techniques, fueled by advancements in DL and neural networks, has urged this revolution, paving the way for innovative applications and solutions.

In recent years, the pervasive integration of ML has extended its reach into the domain of cybersecurity. The significance of ML solutions in detecting and mitigating malicious activities within the intricate landscape of cybersecurity has become increasingly evident. Conventional approaches, primarily relying on static signature-based methods, have proven inadequate in addressing the dynamic and sophisticated nature of contemporary cyber threats. As a result, ML has emerged as a pivotal component in fortifying cybersecurity defenses. Its adaptive and data-driven nature empowers security systems to identify patterns, anomalies, and potential threats in real time, enabling a proactive and responsive approach to cyber threats.

4.2 ML-based anomaly detection

ML is increasingly being used for anomaly-based intrusion detection. The exploration of various algorithms and their combinations is an active area of research, aiming to address the limitations inherent in signature-based IDS.

ML models are built to derive valuable insights from data. These models are designed to perform specific tasks depending on the use case at hand. In the context of anomaly-based IDS, classification, regression, and reconstruction are the most popular and used tasks. The classification task involves the systematic categorization of input data into predefined classes or categories. The primary goal is to assign each input to a specified class, such as "normal or "attack". Regression, or predictive analysis, aims to determine continuous values to estimate an output based on input variables. Finally, reconstruction which is often associated with certain neural networks, involves compressing input data and then reconstructing it, compelling the network to learn efficient feature representations during the encoding and decoding processes.

The ML models used for anomaly-based intrusion detection can be trained in three different manners; supervised, semi-supervised, or unsupervised. Supervised learning requires labeled datasets to train a model, where both inputs and corresponding correct outputs are provided. The model eventually learns to map inputs to outputs, making predictions or classifications based on the learned patterns. ANN, DT, KNN, and bayesian networks are some of the commonly used supervised methods. Supervised learning techniques have well-defined objectives making it easier to evaluate and measure performance. However, they assume the availability of labeled data, with the attacks' type known and data correctly labeled. Whereas, in real-world applications, zero-day attacks never seen in the training dataset, may occur. Moreover, only a limited amount of labeled data is available in reality, with possibly inaccurate annotation. To address these challenges, semi-supervised and unsupervised learning methods are utilized. Semi-supervised learning techniques involve using a combination of labeled and unlabeled data to train the model. This allows for leveraging the information from the labeled data while also taking advantage of the larger amount of unlabeled data. Unsupervised learning, on the other hand, does not require labeled data and can uncover hidden patterns or structures within the data. By allowing the model to learn without predefined labels, unsupervised learning can be effective in detecting anomalies and identifying new types of attacks that have not been seen before. However, it is important to consider that while unsupervised learning can be effective in certain scenarios, it may not provide detailed information about the type of attack. In contrast, semi-supervised learning, as a combination of supervised and unsupervised learning, addresses some of the limitations of both techniques. It leverages a small amount of labeled data to provide detailed information about the type of attack, while also benefiting from the larger amount of unlabeled data to improve overall detection capabilities. Overall, the choice between supervised, semi-supervised, or unsupervised learning depends on the availability of labeled data, the nature of the attacks being studied, and the specific goals and constraints of the analysis.

4.3 DL-based anomaly detection

Numerous ML methods have been proposed for anomaly-based intrusion detection. However, as the volume and complexity of Internet traffic continue to grow, accompanied by the generation of multi-dimensional and large-scale data, and, increasingly sophisticated attack scenarios, traditional shallow ML approaches are deemed insufficient in addressing the evolving security challenges. In response, researchers have turned their attention to DL to explore its potential for intrusion detection.

DL techniques have demonstrated effectiveness in dimensionality reduction and classification tasks, making them ideal for handling the intricacies of network traffic data. Deep networks in DL-based intrusion detection systems learn from historical traffic data, comprising normal and anomalous traffic, without the need for human intervention. Moreover, DL models can automatically extract relevant features from the data and identify complex correlations, enabling them to effectively detect zero-day attacks and complex attack patterns.

DL-based intrusion detection can be achieved using three main architectures, discriminative, generative, and hybrid. Discriminative models leverage labeled data and directly learn to distinguish between normal and anomalous data. They essentially act as anomaly classifiers, assigning a probability score to each data point based on its likelihood of being anomalous. CNN and Long Short-Term Memory (LSTM) are common examples of discriminative architectures. The training process of generative models, on the other hand, involves exposing the model to a diverse set of normal network activities, allowing it to learn the inherent patterns and features and consequently the underlying distribution of "normal" data. Once trained, the model can detect deviations or anomalies by identifying patterns in network data that differ from what it has learned during training, i.e., anything the model finds difficult to reconstruct or generate, is flagged as an anomaly. AEs are a famous example of generative models used for anomaly-based IDS. These models compress the input data into a latent representation and then attempt to reconstruct the original data point from that representation. Anomalies show significant reconstruction errors. Last, hybrid architecture combines the strengths of generative and discriminative approaches. Hybrid models leverage the generative model's ability to capture the normal data distribution with the discriminative model's classification power. A hybrid model may use a GAN to generate synthetic normal data and a CNN to classify incoming data, offering a robust approach that benefits from both generative and discriminative capabilities.

The use of DL in intrusion detection systems has gained significant attention due to its ability to handle the growing complexity and volume of Internet traffic. Additionally, DL models in intrusion detection can perform both feature extraction and classification tasks simultaneously, making them more efficient and accurate compared to shallow ML algorithms. Therefore, researchers have conducted systematic studies to evaluate the use of DL in intrusion detection systems [79].

Generative models, particularly, have been widely used in the field of intrusion detection. For instance, Zavrak et al. [15] proposed a method based on a VAE to detect anomalies in network traffic flows. The reconstruction error was used to classify network flows as anomalous or normal. By conducting experimentation on the CIC-IDS2017 dataset, the results are shown to be similar to AE and OCSVM models in terms of ROC metric. Another work introduced by Azmin et al. [80] combines a Variational Laplace AutoEncoder (VLAE) and a Deep Neural Network (DNN) for intrusion detection. They improved the existing VLAE by incorporating class labels as input to the autoencoder. The new model named Conditional Variational Laplace AutoEncoder (CVLAE) is then used to learn the latent variable while the DNN is employed as a classifier. For instance, Li et al. [81] adopted Wasserstein GAN Divergence (WGAN-DIV) for data generation for each class in the dataset. Then, XGBoost was trained on the enhanced data and was used for detection. The use of VAE and GAN models for anomaly-based intrusion detection has been thoroughly investigated in the literature and has shown promising results. However, both approaches have their advantages and limitations. Many other works have proposed the use of VAE and GAN models for IDS [82, 83, 84, 85, 86]. Normalizing Flow (NF), which are a class of generative models [87, 88, 89] have also been explored in the field of anomaly-based intrusion detection. The authors of this work [90] proposed a semi-supervised learning scheme that leverages normal data with available real anomaly records along with pseudo-anomalies sampled using NF to train a classifier.

4.4 Federated learning for intrusion detection

Traditional centralized intrusion detection methods have long been the cornerstone of network security [91], offering a consolidated approach for monitoring and analyzing network traffic at a central server. This centralized approach involves deploying sensors strategically across the network, to collect data on traffic patterns and behaviors. The collected data is then transmitted to a central server where it undergoes analysis using predefined signatures or anomaly detection algorithms. Although centralized intrusion detection methods have their advantages in terms of streamlined management, a unified view of security events, and high detection accuracy, they also suffer from limitations. First, the centralized approach can create a single point of failure, as an attack on the central server can comprise the entire IDS. Second, with the overwhelming amount of data generated and transmitted to the server, latency can be a major concern, leading to delays in detecting and responding to intrusions. Moreover, the network traffic generated is private in nature, which raises concerns about the privacy implications of transmitting all network traffic to a centralized server. In response to these challenges associated with centralized systems, on-device learning has emerged as an alternative. On-device intrusion detection [92, 93, 94] involves deploying intrusion detection capabilities directly on individual network devices or endpoints. This decentralization of intrusion detection shifts the analysis and decision-making process closer to the source of the network traffic, resulting in faster response times and reducing the need for transmitting sensitive network traffic to a central server. Additionally, on-device learning allows for more granular and context-aware analysis, as it can leverage each device's characteristics. However, on-device learning introduces its own set of challenges. Each device must have sufficient computational resources to perform the necessary analysis and detection tasks. Moreover, these systems need frequent updates to stay effective against the evolving threat landscape. Further, on-device learning exhibits a fundamental constraint by confining its knowledge acquisition to individual user experiences (isolated learning). Consequently, no inter-device knowledge transfer occurs, impeding the detection of recurrent threats across separate devices despite identical behavioral patterns. The inherent isolation of on-device learning models restricts their capacity to exploit the collective power of shared threat intelligence.

Despite their advantages, both centralized and on-device learning paradigms possess fundamental limitations, necessitating the consideration of other learning architectures. A viable solution should strike a balance between privacy, communication cost, latency, and accuracy. In simpler terms, it must ensure data privacy while achieving high model accuracy, with minimal communication cost and acceptable latency. FL has become an increasingly popular approach for intrusion detection [95]. FL addresses the privacy concerns associated with centralized methods while fostering collaboration and knowledge-sharing among devices, offering a more comprehensive and privacy-centric solution for intrusion detection.

4.4.1 Federated learning for network intrusion detection

Among the methods of developing FL-based IDS discussed in the literature, several works have leveraged FL to perform intrusion detection in IoT networks. For instance, The authors of this work [96] have introduced DÏoT, a self-learning distributed system for detecting anomalous behaviors in IoT networks. The proposed solution utilizes FL to efficiently aggregate behavior profiles. The authors evaluated their approach on 30 IoT devices contaminated with real IoT malware and demonstrated that their system can achieve up to 95.6% true positive rate with zero false alarms. Another work by Rahman et al. [93] proposed an FL-based system for intrusion detection in IoT devices to enable knowledge sharing between peers without losing to privacy issues. The evaluation was conducted on the NSL-KDD [97] dataset considering several data distribution scenarios and a comparison between FL, on-device, and centralized approaches was made. Despite the important difference between FL and centralized learning, yet, FL outperforms on-device learning and is the closest to centralized learning. Similarly in this paper [98], the authors proposed DC-Adam, an asynchronous FL anomaly detection approach for IoT systems. For anomaly detection, a denoising autoencoder model was employed, where the reconstruction error was considered as an anomaly score. The proposed approach was evaluated on MNIST [99], CIC-IDS2017 [100], and IoT-23 [101], and was compared with three other approaches, namely Asynchronous Adam (Asyn-Adam), Asynchronous SGD (Asyn-SGD), and Synchronous Adam (Syn-Adam). Asyn-DC-Adam was proven to converge steadily compared to Asyn-Adam and Asyn-SGD and nearly matches Syn-

Adam. Moreover, Asyn-DC-Adam outperforms all the before-mentioned approaches in terms of accuracy, precision, recall, and F1-score.

Despite the benefits of FL approaches especially in IDS, FL models are susceptible to many adversarial attacks leading to their failure. This was shown in [102], where the authors investigated FL for malware detection in IoT devices for both cases supervised (based on a Multi-Layer Perceptron (MLP) model) and unsupervised (based on an Auto-Encoder model) learning. The authors used N-BaIoT [103] for evaluation, a dataset that models the traffic of real IoT devices impacted by malware, and compared the proposed framework with centralized and distributed architectures. High accuracy was obtained (up to 99%) for both supervised and unsupervised learning and under several scenarios, this is mainly because N-BaIoT is considered an easy and less complex dataset. However, this accuracy is shown to drop drastically under adversarial attacks, and though the use of robust aggregation functions, such as median, shows an improvement compared to *FedAvg* yet insufficient, demonstrating the need for more robust countermeasures.

Several other works have been proposed for FL-based NIDS. In fact, the major challenges are related to data issues, such as data scarcity and data dimensionality issues. The work presented by Zhao et al. [104] tackled the first problem of data scarcity by leveraging the multi-task learning paradigm and the heterogeneity of real-world intrusion datasets, and they proposed MT-DNN-FL, a multi-task deep neural network in FL. On the one hand, the proposed method performs multiple tasks simultaneously, namely, network anomaly detection, traffic recognition, and traffic classification. On the other hand, the adoption of FL architecture guarantees user data privacy. The experiments performed on CIC-IDS2017 [100], ISCXVPN2016 [105], and ISCXTor2016 [106] demonstrated the effectiveness of MT-DNN-FL compared to multiple single-task deep neural networks (DNN, k-NN, RF, etc.). Ayed et al. [107] investigated the effectiveness of FL for anomaly detection in NIDS. For this purpose, the CIC-IDS2017 dataset was used, where the proposed experimentation scenario respects the network topology and node characteristics, more specifically, every IP address presented in the dataset was considered as a node in the FL architecture. The authors conducted several experimentation scenarios depending on the client selection strategy and train/test data percentage. The results showed that this method can achieve up to 93% accuracy in some scenarios while preserving data privacy. In this work [108], the data dimensionality challenge was addressed. The authors employed FL to build an anomaly-based network intrusion detection through an on-device sequential learning neural network ONLAD [109]. To tackle the data dimensionality issue, a greedy feature selection algorithm was used to find the set of features that produces the best accuracy according to each device's target attack types. Then, only the devices with the same feature space are gathered and an FL model is created for each group, resulting in multiple global models. Experiments carried out on the NSL-KDD [97] dataset indicate that the best accuracy obtained is 70.4% and an overall improvement is 25.7% Likewise, Tabassum et al. [86] proposed a privacy-preserving FL-based framework, named FEDGAN-IDS. The latter uses a GAN based model in a distributed setup for two reasons, the first for data augmentation and the second to perform binary and multi-class classification. The proposed framework was evaluated on NSL-KDD [97], KDD-CUPP99 [110], and UNSW-NB15 [111], and was compared with FED-IDS, an IDS based on FL and simple neural networks. FEDGAN-IDS is shown to outperform FED-IDS on all datasets in terms of accuracy and convergence rate.

4.5 Limitations and research gaps

While existing ML/DL and FL–based intrusion detection models show promising results in anomaly detection, they are not without their limitations and challenges. These shortcomings need careful consideration when developing and evaluating ML/DL and FL-based approaches for IDS. The key research gaps can be outlined as follows:

- Most of the existing solutions for IDS are designed with the assumption that both labeled normal and attack samples are available, a condition that may not align with real-world scenarios. In reality, labeled attack samples are limited if not completely unavailable. The process of labeling network traffic requires human intervention and usually is very expensive, time-consuming, and sometimes not feasible due to the constantly evolving nature of cyber threats. In addition, it requires extensive domain expertise to accurately label network traffic as normal or malicious, which further complicates the process. Consequently, the deployment of these supervised solutions for real-world applications may be challenging or impractical due to the lack of labeled data for training.
- Many of the current IDS solutions undergo evaluation using outdated datasets like KDD99 and NSL-KDD. These datasets have been widely used in research and benchmarking, however, it has also been shown that they suffer from several limitations and do not accurately represent modern network traffic and at-

tack patterns. Thus any evaluation or comparison of IDS solutions based solely on these datasets may not provide an accurate reflection of their performance leaving certain aspects unexplored and potentially limiting their applicability to diverse real-world situations.

- Numerous researchers place significant emphasis on developing sophisticated models for IDS while often overlooking the importance of the dataset used for training and evaluation. The dataset serves as the foundation upon which the model is trained, and its quality, diversity, and representativeness significantly impact the model's ability to learn meaningful patterns, make accurate predictions, and detect novel threats. Neglecting the dataset can lead to limited generalizability and biased models that perform well in specific instances but struggle with real-world scenarios.
- Often, researchers tend to design intricate models for FL-based IDS, neglecting the challenges associated with the size and complexity of models within this context. The oversight of addressing the implications of deploying large and complex models in a federated setting can have notable consequences, impacting the efficiency, communication costs, and overall performance of FL-based IDS solutions. It becomes crucial to consider the computational and communication overheads associated with deploying sophisticated models across distributed nodes in the FL framework, as these factors play a pivotal role in determining the feasibility and effectiveness of the IDS implementation.

4.6 Summary

In this chapter, we discussed the use of ML/DL approaches and FL framework for intrusion detection and we have reviewed some of the related solutions in the literature. We have shown that these approaches have demonstrated great potential in enhancing the performance and effectiveness of IDS as well as the privacy of the network data. Leveraging ML/DL techniques allows IDS to autonomously learn and adapt to evolving threat landscapes, enabling the detection of both known and novel cyber threats. Additionally, the FL framework offers a privacy-preserving and decentralized approach, addressing concerns related to sharing sensitive data. The exploration of these advanced methodologies aims to contribute to the development of more robust and adaptive intrusion detection systems capable of mitigating the challenges posed by sophisticated and rapidly evolving cyber threats. However, the existing solutions for IDS based on ML/DL and FL frameworks still face some challenges and limitations. These include issues such as generalizability, data quality and availability. The next chapter will address these challenges and propose solutions to further enhance the capabilities of ML/DL and FL-based IDS.

Part II

Contributions

In the second part of this thesis, we delve into the primary contributions made to the field of intrusion detection and FL. This part is organized into three chapters, each elucidating a distinctive facet of the research. The initial chapter introduces the first significant contribution, an autoencoder-based method for distributed NIDS by leveraging FL and anomaly detection. This novel approach harnesses the power of autoencoders as well as the collaborative strength of FL to enhance the efficacy and privacy of intrusion detection systems. The subsequent chapter delves into the second contribution, which is an examination of the effects of different flow timeout values on the performance of ML models in the context of NIDS. The last chapter presents the third contribution, which is a novel FL approach that handles batch statistics in FL when using DL models with normalization layers.

5 Federated Learning for Anomaly-based NIDS

5.1 Introduction

DL has demonstrated its efficacy across various domains, showcasing its transformative capabilities in numerous applications. Its power extends to diverse fields, where DL models, such as neural networks, have proven instrumental in solving complex problems and tasks. From computer vision and natural language processing to healthcare and finance, DL has exhibited a remarkable ability to learn intricate patterns from data, enabling it to make accurate predictions, classify information, and generate valuable insights. The success of DL lies in its capacity to autonomously extract meaningful representations from raw data, adapt to different contexts, and continually improve performance through iterative learning processes.

In the specific context of intrusion detection, DL is becoming a cornerstone, revolutionizing the way cybersecurity professionals identify and combat threats within computer networks. By utilizing DL algorithms, intrusion detection systems can effectively analyze network traffic patterns and identify anomalous behavior that may indicate a potential cyber attack. Unlike traditional rule-based systems that struggle with novel attack tactics, DL models can continuously learn and evolve, keeping pace with the ever-changing threat landscape. This adaptability allows them to detect zero-day attacks and emerging threats before they cause significant damage. Moreover, DL models can go beyond simply reacting to attacks; they can predict potential threats before they occur. This proactive approach enables pre-emptive measures to be taken, minimizing the impact of attacks and safeguarding crucial systems.

5.2 Problem statement

An efficient IDS should, on the one hand, be able to detect intrusions effectively while incurring low false positive rates. On the other hand, it should provide privacy guarantees to protect sensitive client data. To meet the first requirement, one potential approach could be the use of DL models considering their potential to learn complex patterns from raw data. These models require large amounts of data for training. However, a common scenario in many real-world applications is that a limited amount of labeled attack data is available or expensive to obtain, posing challenges to developing robust IDS. Autoencoders (AEs), a specific class of DL models, have shown great promise in network intrusion detection due to their ability to effectively capture and encode complex patterns in traffic data [112, 113]. AEs have the ability to learn only on normal data, building a baseline of normal traffic and recognizing anomalous behavior based on the underlying distribution of normal data. Any deviation from the pre-established baseline is identified as a potential anomaly. This comes with the potential to detect new and sophisticated attacks that might not be identified by traditional signature-based methods. As for the second requirement, FL presents a compelling solution by allowing the training of a global intrusion detection model across decentralized edge devices without centrally storing sensitive data. This decentralized approach addresses privacy concerns, as the model learns from local data without the need for raw data transmission, thereby preserving the confidentiality of client information. FL coupled with AE models, thus holds promise in delivering an effective and privacy-preserving solution for intrusion detection in distributed environments

Most of the proposed solutions, especially those based on AEs, require pooling all network traffic data on a central server for training. While undeniably efficient and accurate, these centralized approaches raise significant privacy concerns and may not be feasible in scenarios where sensitive data needs to remain decentralized [15, 16]. In such scenarios, decentralized approaches and privacy-preserving mechanisms are imperative. Decentralization involves training models locally on individual devices, minimizing the need for centralized data storage. Privacy-preserving techniques, such as FL, offer solutions to ensure data confidentiality during the training process. FL offers a groundbreaking solution, enabling secure and collaborative model training without compromising data privacy. In this chapter, we tackle this issue by leveraging FL and AEs to build efficient and privacy-preserving NIDS.

5.3 Methodology

In this section, we introduce a structured approach to securely train an intrusion detection model in distributed settings. This approach consists of two main parts, namely AE and FL. In the following, we first describe the practice that trains an AE-based model across multiple decentralized clients. Then, we present the proposed method *Fed-ANIDS*, that combines autoencoders with FL to enable secure and privacy-preserving training of an intrusion detection model in distributed settings.

5.3.1 Distributed learning using AE-based models

We assume that there are K clients in the network installed in different locations and connected to a central server. Each of which possesses collected benign data D_k that is kept private. We seek to build a ML-based NIDS leveraging FL, i.e. privately learn a global model from the network data of all clients for network intrusion detection. The FL training requires protecting the privacy of the clients, while the NIDS should accurately detect attacks with a low rate of false alarms.

We design a distributed NIDS using autoencoders (Simple, AE, Variational Autoencoder (VAE), Adversarial Autoencoder (AAE)) and FL framework. Each client connected to the central server runs a copy of the global encoder and the global decoder (and the global discriminator for AAE) with his local benign data. Once the local AE is trained, the updated weights are shared with the central server for aggregation. At the detection phase, the global AE aims to report any possible network intrusion. Figure 5.1 depicts the architecture of distributed learning using AE-based models for NIDS using FL.

At each training round $t \in E$, a subset S_t of $m = max(C \times K, 1)$ clients are selected at random to take part in the current round such that C is the fraction of clients to be chosen to participate. Each client $k \in S_t$ receives the global model and optimizes it with its local data D_i for I local iterations. The optimization is done using the mini-batch gradient descent technique. Once the training is completed, the client k sends the local weights θ_t^i (encoder), ϕ_t^i (decoder), and χ_t^i (discriminator) back to the server for aggregation. These steps are repeated until we reach convergence, i.e. the performance of the model gets closer and closer to a specific value. Note that the server acts solely as a coordinator, ensuring that the aggregated model represents

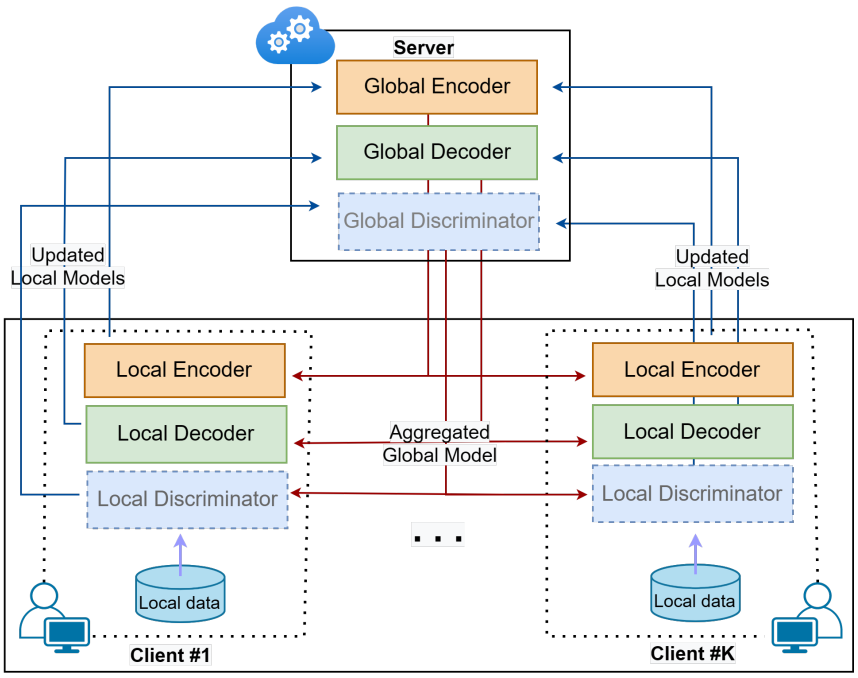


Figure 5.1: Distributed learning of different variations of autoencoders (simple AE, VAE, and AAE) for NIDS using FL.

the collective knowledge of all participating clients without having any access to the data during training. This ensures privacy protection and prevents the server from gathering any sensitive client data during the training process. Table 5.1 introduces the main notations used in the remainder of this chapter.

5.3.2 Proposed method *Fed-ANIDS*

We propose *Fed-ANIDS*, an anomaly-based network intrusion detection method based on FL and AE. Figure 5.2 presents the overall architecture of *Fed-ANIDS*. It consists of four main components: 1) *Global model initialization*, 2) *Local training*, 3) *Model aggregation*, and 4) *Model dissemination*. Each phase is thoroughly explained in the remainder of this section.

5.3.2.1 Global model initialization

We assume that we are in a distributed learning network. The central server starts the learning process by initializing the weights of the global model as well as the hyperparameters needed for the training such as learning rates, momentum, δ value (for *FedProx*), etc. The global model can be an AE, a VAE, or an AAE. In the case of the AAE model, the server is also in charge of defining a prior distribution $p(z)$. Once

Table 5.1: Notations used to describe various steps in the remaining of the chapter.

Notation	Description
K	All clients/entities in the network
E	Total number of communication rounds
I	Total number of local iterations
N	Size of mini-batch
IS	Intrusion score
thr	Anomaly detection threshold
δ	The penalty term for*FedProx* algorithm
D_k	Local dataset of client k
α	Learning rate of Encoder
β	Learning rate of Decoder
γ	Learning rate of Discriminator
$p(z)$	The prior distribution for AAE
θ_t	Encoder parameter at round t
ϕ_t	Decoder parameter at round t
χ_t	Discriminator parameter at round t
$\mathcal{B}$	The local mini-batch size

the initialization is done, the server shares the model and the hyperparameters with the clients selected in the first round.

5.3.2.2 Local model training

We consider K clients that collaboratively train a global model for a network intrusion detection task. The clients perform two main tasks, data preprocessing, and local training:

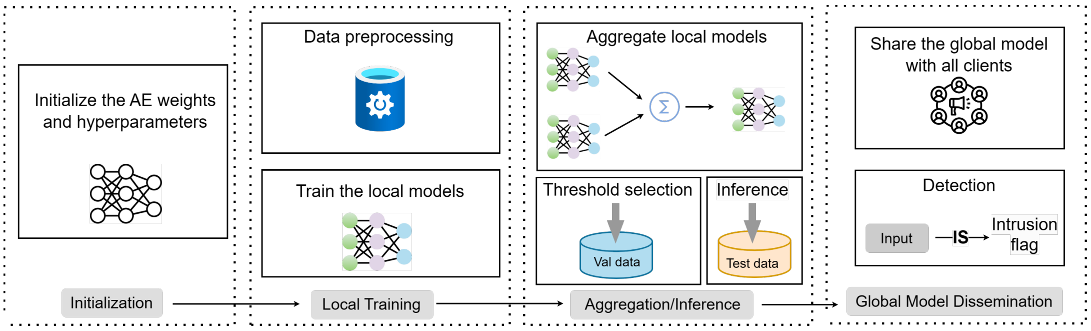


Figure 5.2: *Fed-ANIDS* architecture which consists of 4 main components including, global model initialization, local training, model aggregation, and model dissemination.

Feature extraction & data preprocessing. Before being fed to the AE-based model, each client ensures that his data is prepared and of quality to be consumed. For each dataset, the publicly available *PCAP*[1] files were utilized and preprocessed according to the works of [114, 115]. First, the *pcapfix*[2] tool was run on all *PCAPs* to repair any possible corrupted or damaged ones. The output *PCAP* files are then run through *reordercap*[3], a program that sorts packets by timestamp. This step is most helpful in cases where a file has been created by combining frames from more than one source without taking time order into consideration. The resulting files are fixed and ordered *PCAPs* ready to use. Existing NIDS datasets suffer from various pitfalls related to flow construction, labeling, and attack simulation. Therefore, [114] and [115] have proposed an improved version of the CICFlowMeter tool to generate cleaner NIDS datasets. We use the proposed tool to extract 87 statistical flow features from *PCAP* files and save them into *CSV* files. For CIC-IDS2017 and CSE-CIC-IDS2018 flow labeling, we follow the guidelines[4] of [115]. The extracted features are composed of both numerical and categorical features, hence we convert character data into numeric values (features encoding). Finally, we perform min-max scaling to normalize the data. Figure 5.3 illustrates the data preprocessing pipeline.

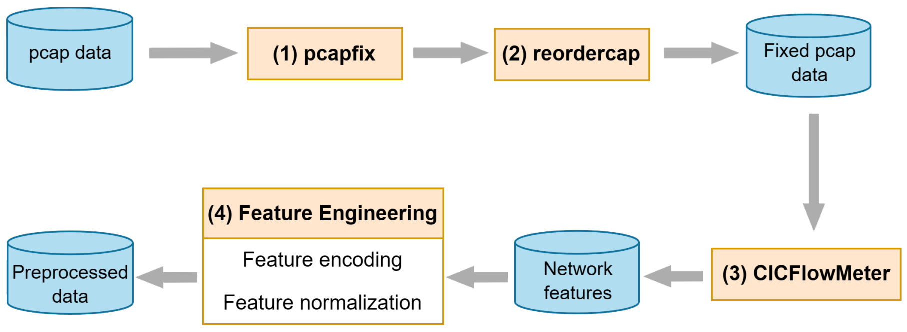


Figure 5.3: Data preprocessing pipeline.

Local training. If selected, a client downloads the current global state of the AE from the server. The AE is then fed with only normal samples of the client's local data and trained to learn the encoding and the reconstruction of the normal behavior. The mean squared error is minimized between the input x and its reconstruction x'. To train the discriminator of the AAE, an isotropic gaussian distribution $\mathcal{N}(\mu, I)$ is used as the imposed prior. *Fed-ANIDS* uses the *FedProx* algorithm to update the local

[1]Packet Capture, a file format for storing raw network packets.
[2]`https://github.com/Rup0rt/pcapfix`
[3]`https://www.wireshark.org/docs/man-pages/reordercap.html`
[4]`https://github.com/GintsEngelen/CNS2022_Code`

models, hence a regularization term $\frac{\delta}{2}\left\|w - w_t\right\|^2$ is added to the loss function of each client, where w are the parameters of the global model, w_t are the parameters of a local model at round t, and $\delta \in [0, 1]$ is a regularizer factor. The aforementioned steps are shown in Algorithm 1 for the client's side.

Algorithm 1 CLIENTUPDATE: optimize clients models weights

```
Input: The parameters $\theta$, $\phi$, and $\chi$ for the encoder, decoder, and discriminator resp; the prior distribution; the local normal data $X = [x_0, ..., x_{N-1}] \sim p(x)$; the number of local epochs $I$; the set of mini-batches $\mathcal{B}$ each of size $N$; parameters $\alpha, \beta, \gamma$ for the learning rates for the encoder, decoder, and discriminator resp; the regularizer factor $\delta$ FedProx.
Output: Normal or Attack.
for $i = 1$ to $I$ do
    for batch $b \in \mathcal{B}$ do
        ▷ Reconstruction phase
        ▷ Minimize the reconstruction loss
        $\mathcal{L}_{rec} = \frac{1}{N}\sum_{i=0}^{N-1}(x_i - x_i')^2 + \frac{\delta}{2}\left\|\phi - \phi_t\right\|^2$
        $\phi \leftarrow \phi - \alpha \nabla_{\phi} \mathcal{L}_{rec}$
        $\theta \leftarrow \theta - \beta \nabla_{\theta} \mathcal{L}_{rec}$
        if AAE then:
            ▷ Generate fake samples $z_{fake} \sim q_{\phi}(z|x)$ by the encoder
            ▷ Draw samples from the prior $z_{real} \sim p(z)$
            ▷ Regularization phase
            ▷ Train the discriminator
            $\mathcal{L}_{dis} = -\frac{1}{N}[\sum_{i=0}^{N-1} log D_{\chi}(z_{real_i}) + \sum_{i=0}^{N-1} log(1 - D_{\succ}(z_{fake_i}))] + \frac{\delta}{2}\left\|\chi - \chi_t\right\|^2$
            $\chi \leftarrow \chi - \gamma \nabla_{\chi} \mathcal{L}_{dis}$
            ▷ Train the generator (encoder) to match the prior
            $\mathcal{L}_{prior} = \frac{1}{N}\sum_{i=0}^{N-1} log(1 - D_{\chi}(z_{fake_i})) + \frac{\delta}{2}\left\|\chi - \chi_t\right\|^2$
            $\phi \leftarrow \phi - \alpha \nabla_{\phi} \mathcal{L}_{prior}$
        end if
    end for
end for
▷ Send $\theta$, $\phi$, and $\chi$ to the server
```

5.3.2.3 Model aggregation

Model aggregation is a crucial step in FL. It allows the local models trained on separate devices to be combined into a single global model while maintaining data privacy. At the end of each communication round, each participating client sends his local update of the model to the server. Upon receiving all the updates, the server computes the weighted average as the new global weight parameters. The global model is then updated with the weighted average, and the process is repeated until convergence. Algorithm 2 depicts the steps of model aggregation executed by the server.

5.3.2.4 Threshold selection

Once the training procedure is completed, a score threshold must be computed for the intrusion detection phase. For this purpose, we set a separate validation set upon which we determine a proper threshold for the global model. We should mention that the model performance is heavily reliant on the threshold value. On the one hand, a higher value would result in fewer false alarms but could also result in marking more attacks as normal instances. On the other hand, a lower value would produce more false alarms and mark more normal instances as attacks. We propose to compute the threshold *thr* following Equation 5.1, i.e., the threshold is the sum and standard deviation of MSE over the validation set [116].

$$thr = \overline{MSE(D_{val}, \phi_t))} + s(MSE(D_{val}, \phi_t)) \tag{5.1}$$

The determined threshold is then used at the inference stage. We compare the Intrusion Score (IS), which is the reconstruction loss, with the pre-computed threshold. If the intrusion score is greater than the threshold, then the instance is considered an intrusion; otherwise, the instance is benign.

Algorithm 2 SERVERAGG: build global model by aggregating clients models

```
Input: $\theta$, $\phi$, and$\chi$ the parameters for the encoder, the decoder, and the discriminator resp; the $K$ clients; the number of global rounds $E$;
Output: $\theta$, $\phi$, and $\chi$
Initialize $\theta_0$, $\phi_0$, and $\chi_0$
for each round $t = 0, \ldots, E-1$ do
    m ← max(C × K, 1)
    $S_t$ ← select a random set of m clients
    ▷ send $\theta_t$, $\phi_t$ and $\chi_t$ to each client k ∈ $S_t$
    for each client k ∈ $S_t$ do
        $\theta^k_{t+1}, \phi^k_{t+1}, \chi^k_{t+1} \leftarrow clientUpdate(w_t)$
    end for
    $\theta_{t+1} = \frac{1}{K} \sum_{k \in S_t} \theta^k_{t+1}$
    $\phi_{t+1} = \frac{1}{K} \sum_{k \in S_t} \phi^k_{t+1}$
    if AAE then
        $\chi_{t+1} = \frac{1}{K} \sum_{k \in S_t} \chi^k_{t+1}$
    end if
    Disseminate the updated parameters for the next round.
end for
```

5.3.2.5 Model parameters dissemination

Once the training is completed and the convergence is reached, the latest update of the global model is shared with all clients available in the network. When a new instance X_{new} comes into the network, an intrusion score is computed (reconstruction loss). The instance X_{new} is marked as an intrusion if the IS is greater than the threshold, else it is marked as a normal instance.

$$X_{new} = \begin{cases} Anomaly, & \text{if } IS > thr \\ Normal, & \text{otherwise} \end{cases} \tag{5.2}$$

5.4 Experiments and results

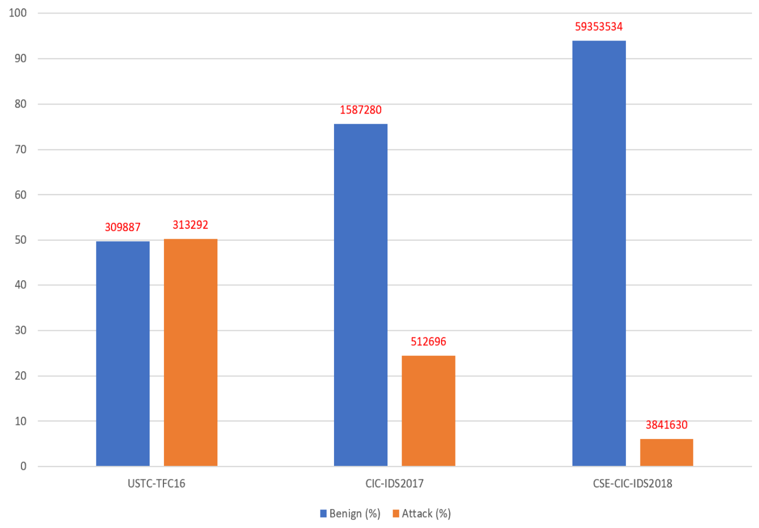


Figure 5.4: The distribution of benign and attack samples of USTC-TFC2016, CIC-IDS2017, and CSE-CIC-IDS2018 datasets.

In this section, we evaluate the performance of the proposed method *Fed-ANIDS* with various well-known datasets (USTC-TFC2016, CIC-IDS2017, and CSE-CIC-IDS2018). We mainly focus on answering the following research questions:

RQ1: How does *Fed-ANIDS* perform in comparison to other baselines (Generative Adversarial Network (GAN), and a Bidirectional GAN (BiGAN)) according to various intrusion detection metrics?

RQ2: What is the most efficient distributed algorithm when comparing *FedProx* and *FedAvg*?

RQ3: What is the performance of *Fed-ANIDS* when training data comes from various environments, and how does the proposed method perform with previously unseen data?

5.4.1 Datasets

We consider three well-known datasets, namely, USTC-TFC2016 [28], CIC-IDS2017 [29], and CSE-CIC-IDS2018 [5]. We split USTC-TFC2016 at a 70-20-10 ratio for training, validation, and test sets. For CIC-IDS2017, we pick the first four days of data for training and validation (Monday through Thursday) and we test on Friday data. Since there are two weeks of data in CSE-CIC-IDS2018, we allocate the first week for training and validation, while we preserve the second week for testing.

In real-world scenarios, normal data in network intrusion detection is generally more abundant than attack samples due to the nature of network traffic. Considering the prevalence of normal network traffic, training an anomaly-based intrusion detection system solely on normal data is a practical approach. Therefore, in this work, before partitioning the data over clients, only the normal samples are left in the training and validation sets. In contrast, the remaining attack samples are included in the test set. Finally, the training set is equally and randomly partitioned between all the clients in the system, ensuring a fair distribution of normal data for training. Figure 5.4 depicts the distribution of benign and attack samples of each dataset.

5.4.2 Experimental settings

In this work, we consider a distributed environment of $K = 10$ clients in total. We fix the client fraction at $C = 0.5$. We set the local iterations at $I = 10$ and the global communication rounds at $E = 30$ epochs for USTC-TFC2016 and CIC-IDS2017 and $E = 10$ for CSE-CIC-IDS2018. For training hyperparameters, we use the validation set to find the best combination that maximizes the performance. For the regularization parameter for *FedProx*, we tested the values in the candidate set {0.1, 0.01, 0.001, 1e-4, 1e-5, 1e-6}. As for the learning rate and weight decay, we tested the values in the range {0.01, 0.001, 1e-4, 1e-5} and {1e-4, 1e-5, 1e-6} respectively.

The AE architecture adopted in this work consists of two components, namely an encoder and a decoder, which are modeled as two fully-connected layer networks specified by the following number of neurons per layer (87-64-32) and (32-64-87), respectively. Each layer is followed by a ReLU activation function. Similarly, the VAE

[5]`https://www.unb.ca/cic/datasets/ids-2018.html`

Table 5.2: Performance evaluation of *Fed-ANIDS* with USTC-TFC2016 dataset.

	Fed-ANIDS(ours)						Baseline			
	AE		VAE		AAE		GAN		BiGAN	
Metrics	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*
F1-score (%)	99.35	99.44	99.79	99.77	99.93	**99.94**	94.96	96.77	61.02	72.78
FDR (%)	0.46	0.22	0.39	**0.01**	0.18	0.18	9.90	8.45	5.14	15.43
Accuracy (%)	99.54	99.60	99.85	99.84	99.94	**99.95**	96.38	97.66	81.40	84.68

Table 5.3: Performance evaluation of *Fed-ANIDS*with CIC-IDS2017 dataset.

	Fed-ANIDS(ours)						Baseline			
	AE		VAE		AAE		GAN		BiGAN	
Metrics	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*
F1-score (%)	92.51	**92.73**	60.96	62.09	80.11	77.17	80.28	74.95	71.24	78.12
FDR (%)	1.75	1.69	49.49	45.66	**0.34**	5.60	24.68	34.99	30.39	34.96
Accuracy (%)	93.36	**93.54**	64.34	66.56	83.94	81.62	81.90	76.28	74.97	78.63

Table 5.4: Performance evaluation of *Fed-ANIDS*with CSE-CIC-IDS2018 dataset.

	Fed-ANIDS(ours)						Baseline			
	AE		VAE		AAE		GAN		BiGAN	
Metrics	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*
F1-score (%)	88.48	86.41	89.40	**90.64**	45.51	45.52	47.85	46.60	55.01	55.12
FDR (%)	1.51	**0.60**	2.14	1.81	16.67	16.66	17.13	16.85	14.75	12.92
Accuracy (%)	92.90	91.08	93.72	**94.48**	83.34	83.34	75.58	80.77	72.95	73.36

architecture consists of four layers such that (87-64-64-32) and (32-64-64-87) are the number of neurons per layer for the encoder and the decoder, respectively. For AAE, we model both the encoder and the decoder as three fully-connected layer networks specified by the following number of neurons per layer (87-16-4-2) and (2-4-16-87), respectively. Each layer is followed by a LeakyReLU activation function. Similarly, the discriminator consists of three fully-connected layers specified by the following number of neurons per layer (16-4-2). Each layer is followed by a LeakyRelu activation function except the last layer which is followed by a sigmoid function.

To evaluate the performance of our approach, we consider three standard intrusion detection metrics, F1-score, Accuracy, and False Discovery Rate (FDR). Note that FDR represents the ratio of falsely reported anomalies to total anomalies. We use Python and ML libraries such as PyTorch, Numpy, and Pandas to perform all experiments. This work was carried out using the African SuperComputing Center HPC service, supported by Mohammed VI Polytechnic University[6].

5.4.3 Performance evaluation

To make our evaluations credible, we compare *Fed-ANIDS*with prior works that have a similar context. [86] introduced a system that leverages AD and FL for intrusion detection. The proposed system is mainly based on GANs [57] which are commonly employed across diverse fields, including computer vision and anomaly detection.

According to our proposed method, we implemented two baseline models and evaluated their performances with USTC-TFC2016, CIC-IDS2017, and CSE-CIC-IDS2018. The first model is a simple GAN while the second model is a BiGAN [117]. The latter has the ability to learn rich representations in applications like unsupervised learning and anomaly detection [118]. Both Autoencoders and GANs are unsupervised learning techniques that can be used for network intrusion detection. The two other major distinctions between our work and [86] are 1) we use the *FedProx* FL technique, unlike the work of [86] which uses the *FedAvg* FL technique and 2) we perform all evaluations with cleaner versions of flow features extracted from USTC-TFC2016, CIC-IDS2017, and CSE-CIC-IDS2018 datasets, while [86] have conducted their experiments on images generated from NSL-KDD[7], KDD [97], and UNSW-NB15 [111] datasets. The format of datasets is important in an FL network. Tabular datasets that contain various features extracted from raw traffic are more interpretable, require less storage, and result in a smaller global model size when compared to raw traffic image datasets. Considering that the edge entities in FL systems usually have limited resources, we work with flow features-based datasets.

Table 5.2 provides the obtained results when evaluating our model with the USTC-TFC2016 dataset. We observe that AAE with the *FedProx* algorithm yields the best results with the highest F1-score of 99.94% and accuracy of 99.95% and the smallest FDR value of 0.18%, followed by VAE and AE which also perform well. On the contrary, GAN and BiGAN perform poorly, especially BiGAN which induces a high FDR with a low F1-score and accuracy. In addition, *FedProx* is always on par or better than *FedAvg* for all models. Table 5.3 presents the evaluation performance with CIC-IDS2017. We observe that a simple AE trained with *FedProx* outperforms all other models, with an accuracy that exceeded 93% and the lowest FDR of 1.693%. Lastly, Table 5.4 gives the scored results when using the CSE-CIC-IDS2018 dataset. VAE is found to be more effective in terms of F1-score and accuracy. The simple AE model

[6]`https://ondemand.hpc.um6p.ma/`

[7]`http://nsl.cs.unb.ca/NSL-KDD/`

achieved the best FDR 0.6% at the cost of a drop in F1 score and accuracy. Overall, the simple AE model is the best-performing model.

For each dataset, at least one of the autoencoder-based approaches with *FedProx* outperforms other GAN-based models. Thus, our findings suggest that in the context of NIDS, autoencoders are more effective in detecting potential threats compared to GAN-based approaches. Therefore, solutions for network intrusion detection in real-world scenarios that are based on autoencoders would be more appropriate than GAN-based solutions since they are simple, light, and computationally efficient.

5.4.4 Model generalization for heterogeneous networks

We also evaluate the generalization performance of *Fed-ANIDS* according to three scenarios using the three previously presented datasets (which are collected from different sources and environments). The first scenario consists of training the model using a subset of USTC-TFC2016 and CIC-IDS2017 datasets. The second scenario builds the model with USTC-TFC2016 and CSE-CIC-IDS2018 datasets. The last scenario optimizes the model with CIC-IDS2017 and CSE-CIC-IDS2018. For each scenario, we evaluate the model with a subset of the testing set of all datasets, such that, for each scenario one of the datasets was never seen before. These scenarios allow us to test the generalization strength of the proposed model under FL in two ways: 1) how well the model generalizes when training with two datasets collected in different environments, and 2) how the model performs with a new dataset never seen before. Considering a network of twenty clients in total, we select the simple autoencoder (AE) as the autoencoder-based model and we split two datasets among ten clients each (e.g. USTC-TFC2016 and CIC-IDS2017) then we train the model for $E = 30$ rounds such that at each round third of the clients are chosen randomly ($C = 0.3$). Once the training is completed, we evaluate the model not only on the two datasets used for training but also on a third one never seen before (e.g. CSE-CIC-IDS2018).

Table 5.5: Baselines testing results of a simple AE trained on USTC-TFC2016, CIC-IDS2017 and CSE-CIC-IDS2018 datasets seperatly.

	USTC-TFC2016		CIC-IDS2017		CSE-CIC-IDS2018	
Metrics	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*
F1-score (%)	99.35	99.44	68.00	68.16	79.91	89.46
FDR (%)	0.47	0.23	0.14	0.14	1.03	1.91
Accuracy (%)	99.54	99.60	93.72	93.79	84.55	93.01

Prior to the generalization evaluation of Fed-ANIDS, Table 5.5 presents the baseline results of the simple AE model when training separately on each dataset. Tables 5.6, 5.7 and 5.8 provide the evaluation results for the three scenarios of generalization. Compared to the baseline (Table 5.5), we observe that in most cases the performance deteriorates in terms of all metrics and the model cannot generalize on seen and unseen datasets. However, the decrease in performance is much larger for some cases than others. For instance, when considering the first scenario (see Table 5.6), on the one hand, the performance dropped drastically for USTC-TFC2016 despite it being used for training. On the other hand, the model could generalize relatively well on CSE-CIC-IDS2018 which was never seen during training. On the contrary, in the second scenario (see table 5.7) the model could not generalize on the CIC-IDS2017 dataset, which was never seen, yet it performed well on USTC-TFC2018 and CSE-CIC-IDS2018 datasets used for training. Lastly, in table 5.8, the model performs well on the CSE-CIC-IDS2018 dataset but achieves poor results on USTC-TFC2016 and CIC-IDS2017. These results can be attributed to the difference in the dataset size, where CSE-CIC-IDS2018 is almost five times bigger than USTC-TFC2016 and CIC-IDS2017. In view of all the obtained results, the *FedProx* algorithm yields much better results compared to *FedAvg*, in some cases up to 10% F1-score and accuracy and more than 6% FDR. This can be explained by the fact that the *FedProx* algorithm tackles the heterogeneity challenge in federated networks.

Table 5.6: Testing results of a simple AE trained on USTC-TFC2016 and CIC-IDS2017.

	Seen Datasets				Unseen Dataset	
	USTC-TFC2016		CIC-IDS2017		CSE-CIC-IDS2018	
Metrics	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*
F1-score (%)	71.51	78.79	58.12	66.02	64.25	65.14
FDR (%)	41.04	31.25	0.34	0.32	0.078	0.08
Accuracy (%)	72.36	79.80	87.52	93.24	67.38	68.40

Table 5.7: Testing results of a simple AE trained on USTC-TFC2016 and CSE-CIC-IDS2018.

	Seen Datasets				Unseen Dataset	
	USTC-TFC2016		CSE-CIC-IDS2018		CIC-IDS2017	
Metrics	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*
F1-score (%)	80.53	89.27	74.51	85.44	48.69	58.18
FDR (%)	7.98	1.25	3.63	2.41	0.06	0.14
Accuracy (%)	83.72	90.70	80.16	89.95	72.81	86.76

Table 5.8: Testing results of a simple AE trained on CIC-IDS2017 and CSE-CIC-IDS2018.

	Seen Datasets				Unseen Dataset	
	CIC-IDS2017		CSE-CIC-IDS2018		USTC-TFC2016	
Metrics	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*	*FedAvg*	*FedProx*
F1-score (%)	45.47	49.83	80.09	85.87	66.41	67.68
FDR (%)	1.31	1.18	5.28	4.87	42.07	43.93
Accuracy (%)	71.32	80.00	86.45	91.13	69.4	69.16

5.4.5 Distributed vs centralized learning for Anomaly-based NIDS

This section aims to compare the performance of Fed-ANIDS, which is based on FL, with a centralized learning setting. Centralized learning requires all the data to be transferred to a central server for training, meaning that no computation is carried out by the clients. We conducted experiments on the three previously mentioned datasets for each autoencoder variant (AE, VAE, and AAE). For the centralized setting, all data was preprocessed by the central entity, which is also the one in charge of training and testing. Figure 5.5 illustrates the performance comparison between *Fed-ANIDS*and centralized learning according to F1-score, accuracy, and FDR. In terms of both the accuracy and the F1-score, the results show that *Fed-ANIDS*achieves comparable or better performance than centralized learning. Specifically, the centralized performance is generally reached by *Fed-ANIDS*and sometimes even surpassed. Similarly for FDR, *Fed-ANIDS*mostly scores lower FDR than centralized learning. This demonstrates the potential of usingFL for anomaly-based NIDS that successfully reaches or exceeds the centralized performance while preserving the privacy of clients. Furthermore, the use of FL in such systems also reduces the latency and bandwidth consumption since the data is processed locally on each client's device rather than being sent to a central server.

5.5 Summary

In this chapter, we introduced *Fed-ANIDS* an AE-based approach that incorporates FL and anomaly detection for intrusion detection in distributed networks. *Fed-* efficiently enables secure model training and robust intrusion detection by leveraging the benefits of FL and those of autoencoders. We have built the proposed method with three variants of autoencoders, namely, simple AE, VAE, and AAE, and we adopt *FedProx* as the FL algorithm. With a series of experiments and evaluations conducted on three well-known flow-based datasets (USTC-TFC2016, CIC-IDS2017,

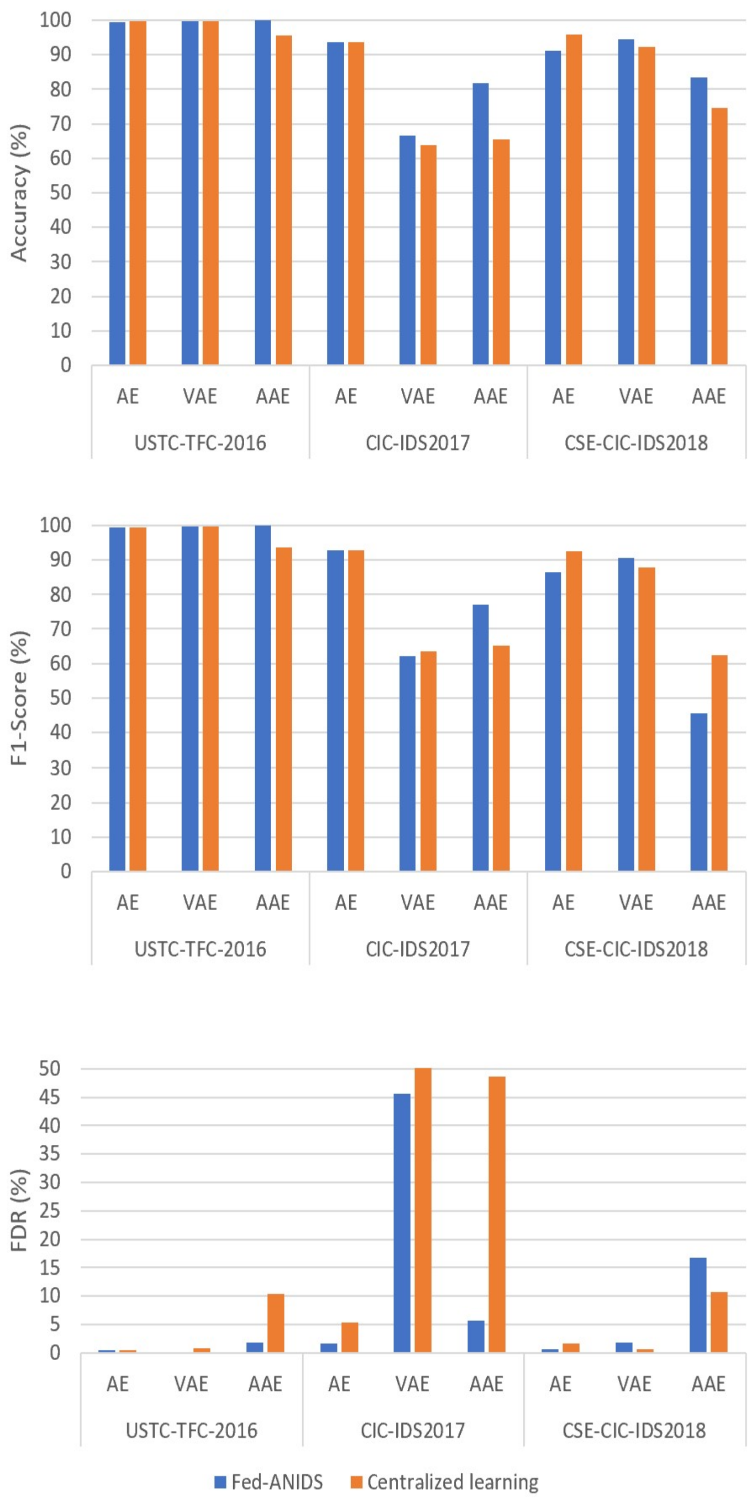


Figure 5.5: Comparion between *Fed-ANIDS*and centralized learning.

and CSE-CIC-IDS2018), we have demonstrated the effectiveness of the proposed method in detecting network intrusions with high accuracy and low false alarms while preserving privacy. Additionally, *Fed-ANIDS* significantly outperforms other GAN-based models. We also show that the *FedProx* algorithm adopted in *Fed-ANIDS* is always on par or better than *FedAvg*.

6 Flow Timeout and the Performance of ML Models for NIDS

6.1 Introduction

In today's era of high-speed networks and massive data volumes, network security and cybersecurity technologies and systems such as NIDS, Network Security Monitoring (NSM), and Industrial Control Systems (ICS) are increasingly shifting their focus from payload-based approaches to a growing reliance on flow-based methods. The substantial volume of network traffic makes the examination of each packet impractical and resource-intensive. Thus flow-based methods [119, 120] emerge as a practical and highly efficient alternative. Unlike payload-based approaches [121], which involve inspecting the complete content of packets, including both headers and payload, flow-based methods leverage aggregated network information in the form of flows [122]. This approach substantially reduces the volume of data requiring analysis, thereby minimizing the time and resources needed for detecting and responding to potential threats.

Integrating ML techniques into flow-based approaches for Network Security Systems (NSS) enhances the efficacy of these tools. Through the utilization of ML algorithms, the analysis of network flows not only gains precision but also adapts to the evolving nature of threat landscapes. This integration plays a pivotal role in significantly reducing the volume of data that requires analysis, thus minimizing the time and resources essential for detecting and responding to potential threats. Yet, it is crucial to emphasize that the effectiveness of these models is tied to the quality of the flow features used in the learning process [17]. Choosing and extracting pertinent features from network traffic data is crucial for the overall performance and accuracy of these ML-based NIDS solutions. Consequently, the feature extraction process significantly impacts these systems' capabilities to identify and effectively mitigate emerging threats. To extract network flow features, several tools are available such as NFStream [12], CICFlowMeter[1], and nProbe[2]. These tools are vital for collecting and analyzing network flows, generally relying on a set of hyperparameters that configure and govern various aspects of the feature extraction process. These hyperparameters influence how features are selected, extracted, and represented from the raw network flow data, thus impacting the quality of the extracted features. One of these properties is the time interval, comprising two timeouts, an active and an idle (inactive) timeout [13]. Modifying the values of these two parameters leads to the extraction of different features, potentially influencing the performance of ML models that rely on these features for their learning and decision-making processes.

6.2 Problem statement

Network analysis software and IDS, both open-source (such as Snort [3] and Suricata [4]), and commercial (such as Palo Alto Networks [123] and Fortinet [124]) incorporate configurations for both active and idle timeouts. While many of these solutions come with predefined and optimal values for these parameters, it is important to note that there are instances where these values may not align with the specific requirements of the network. Opting for a value that is too low can heighten sensitivity to minor network delays. On the other hand, choosing an excessively high session timeout may result in delayed detection of failures. Moreover, when constructing NIDS datasets from the flow data, the quality and relevance of the extracted features highly depend on the idle and active timeouts. Therefore flow timeouts hold substantial importance in the development and assessment of network IDS.

Throughout our literature review, we have noticed that the prevalent approach to configuring flow timeouts involves either adhering to the pre-configured values for both idle and active timeouts throughout the feature extraction process or opting to set new ones without providing substantial justification for their selection [125, 126]. Surprisingly, limited attention has been devoted to investigating the potential ramifications of altering these parameter values on the quality of extracted features and

[1]`https://github.com/ahlashkari/CICFlowMeter`
[2]`https://www.ntop.org/guides/nprobe/cli_options.html`

the subsequent impact on ML models' performance. This chapter seeks therefore to bridge this notable knowledge gap. Its primary goal is to thoroughly investigate the effect of employing different values for both idle and active timeouts while utilizing various ML models. By doing so, we shed light on whether there is a compelling value for these two parameters before the feature extraction phase. Such insights are critical in optimizing NIDS systems and enhancing their ability to detect and respond to emerging security threats.

6.3 Design of experiments

In this section, we delve into the core elements of our experimental design. First, we present an in-depth exploration of both active and idle timeouts, shedding light on their direct impact on feature extraction and, subsequently, on the overall performance of ML models. Following this, we present the feature extraction pipeline employed throughout our study. Then, we provide a brief description of the ML models leveraged in our research. Finally, we discuss the FL setting adopted to assess the performance in a decentralized environment for heterogeneous NIDS. Figure 6.1 summarises the overflow of the proposed design of experiments. The series of experiments conducted in this work, aim to answer the following research questions:

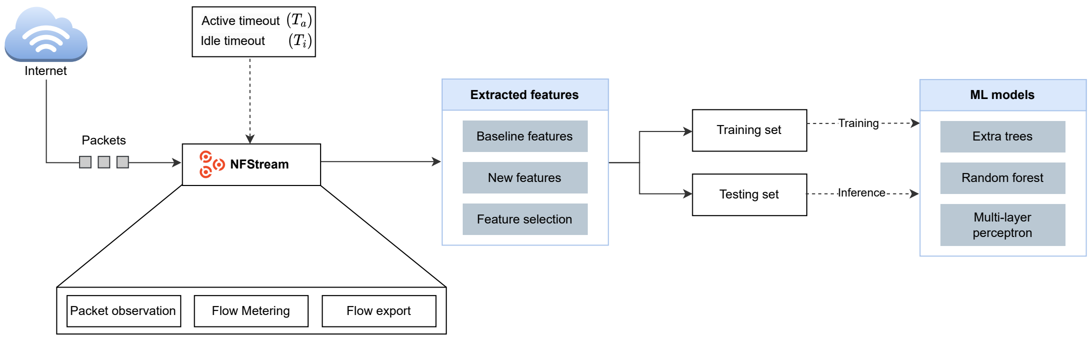


Figure 6.1: The overflow of the proposed design of experiments.

- **RQ1:** What is the impact of idle and active timeouts on the performance of three well-known ML models?
- **RQ2:** Within diverse feature sets of network flows, does a particular combination of idle and active timeouts exhibit superior performance compared to others?

- **RQ3:** In the context of a real-world scenario involving multiple NIDS with varying idle and active timeout settings, is it possible to construct a global model with federated learning that delivers strong performance across all these diverse environments?

- **RQ4:** Taking into account the responses to the preceding research questions, can we provide recommendations regarding the utilization of specific idle and active timeouts in the context of ML-based NIDS?

6.3.1 Feature extraction

In this work, we use NFStream [12] to extract features from raw traffic packets. Our selection was guided by the following considerations: 1) NFStream framework is open-source and enables the analysis of network traffic flow with high throughput on standard hardware. 2) NFStream can identify the applications that generate network traffic flows. This feature, also known as application awareness, is achieved by integrating the nDPI library[3]. 3) NFStream uses parallelism to achieve high-speed live network traffic measurement. 4) NFStream supports tunnel decoding (GTP, CAPWAP, and TZSP), which allows it to accurately identify packets that originate from different flows. 5) NFStream is a flow-based measurement framework that can be used with ML libraries such as TensorFlow [127] and PyTorch [128]. 6) NFStream is extensible, making it possible to create and extract new flow features with just a few lines of Python code.

Furthermore, we build a set of flow features based on four distinct sources, including the features proposed by NFStream [12], the features selected in the work [129], the features introduced in the open source framework [130], and features developed by our team. Table 6.1 presents the set of features created by our team and their descriptions. In addition, we provide in the appendix 8.3 the list of features presented in both works [129] and [130]. The standard features of NFStream [12] can be found in the documentation page [4].

6.3.2 Model training

Empirical Risk Minimization (ERM) aims at finding model parameters that optimize the empirical risk, generally quantified by a loss function. However, the quality and

[3]https://www.ntop.org/products/deep-packet-inspection/ndpi/
[4]`https://www.nfstream.org/docs/api#nflow`

Table 6.1: Set of features developed by our team.

Feature	Description
tcp_init_ms	Time in ms between first SYN and SYN-ACK packet
tcp_synack_ack_ms	Time in ms between SYN-ACK packet and its acknowledgment
tcp_half_closed_time_ms	Time in ms that connection is half closed
num_pkts_after_termination	The number of packet received in 15s after the four-way handshake
bidirectional_transport_bytes	Total bytes of exchanged transport segments
bidirectional_payload_bytes	Total bytes of exchanged packets payload
src2dst_transport_bytes	SRC2DST total bytes of exchanged transport segments
src2dst_payload_bytes	SRC2DST total bytes of exchanged packets payload
dst2src_transport_bytes	DST2SRC total bytes of exchanged transport segments
dst2src_payload_bytes	DST2SRC total bytes of exchanged packets payload

representativeness of the data are paramount, if the data used during the optimization process is biased or noisy, the model's likelihood of scoring optimal results is low, regardless of the algorithm employed. As the task at hand is a multi-class classification, we used three well-known ML classifiers (different learning algorithm classes) to study the impact of *idle* and *active timeouts* on network intrusion detection performance, namely, ET, RF, and MLP classifiers.

6.3.3 Distributed learning in multiple timeouts environment

To assess the potential of distributed learning within the context of multiple NIDS, each characterized by a particular set of idle and active timeouts, we've opted to utilize the federated learning framework. Federated learning is a decentralized ML approach where a global model is collaboratively trained across multiple decentralized devices or data sources, all while keeping data localized. This approach offers advantages, including enhanced privacy, scalability, and efficiency. Our decision to employ federated learning aims to explore its potential benefits within the context of our specific learning task of heterogeneous network intrusion detection systems. By leveraging this decentralized approach, we aim to gain a deeper understanding of its applicability and effectiveness in managing diverse NIDS with varying configurations. Moreover, we seek to uncover whether federated learning can deliver superior results in terms of threat detection compared to traditional centralized approaches.

In our simulation of the distributed setting, we model it as a system consisting of K clients (NIDS) and a central server. Each client possesses data collected with a particular idle and active timeout $(T_{a,k}, T_{i,k})$. During each communication round, a fraction C of clients is selected randomly to collectively optimize the global model. Initially, the server initializes and disseminates the global model to the selected clients in the first round. Subsequently, each client individually optimizes the model using its local data for E local epochs and then sends the model's weight updates back to the server. Once all the updates are received from the participating clients, the server aggregates the local models and updates the global model. The new global model is then shared with the participating clients in the subsequent round. These steps are repeated until convergence is reached, i.e., the loss reaches a steady state. Figure 6.2 depicts the federated learning setting.

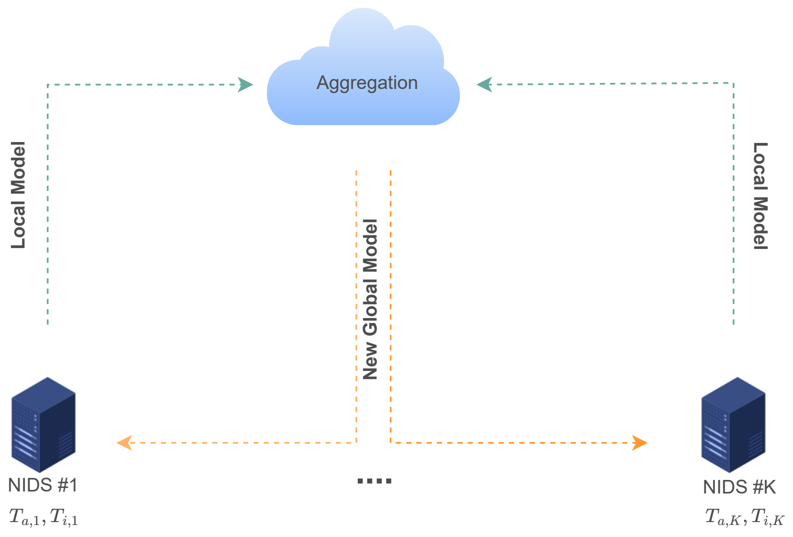


Figure 6.2: Distribued learning using federated learning.

6.4 Experiments and results

This section provides the implementation details and discusses the performance evaluation across four well-known datasets using three ML classifiers. The evaluation is examined from four perspectives: timeout-based, model-based, feature-based, and distributed learning performances.

6.4.1 Implementation details

To evaluate the proposed method in the centralized setting, we consider three classifiers defined in the Scikit-learn library [131]. Extra Trees Classifier (ETC), Random Forest Classifier (RFC), and Multilayer perceptron (MLP). The ETC and RFC models used are both constructed of 100 decision trees. Whereas the MLP model is trained for 1000 iterations with the default parameters. Each model is trained and evaluated on four well-known and public datasets, USTC-TFC2016, UNSW-NB15, CUPID, and

CIC-IDS2017. For each dataset, we allocate two-thirds of the total size of the data to the training and one-third to the testing.

For the distributed scheme, we consider an environment of 32 clients, each owning local data extracted using a particular and unique idle and active timeout couple. We adopt the standard FL algorithm, *FedAvg* [58]. At each communication round a subset of fraction $C = 0.5$ is selected to optimize the global model on their local data for $E = 5$ epochs. The process is repeated for $R = 15$ communication rounds for the USTC-TFC2016 dataset and $R = 10$ for the CUPID, CIC-IDS2017, and UNSW-NB15 datasets. We define the global model as an MLP neural network consisting of three fully connected layers specified by the following number of neurons per layer (57, 32, 16).

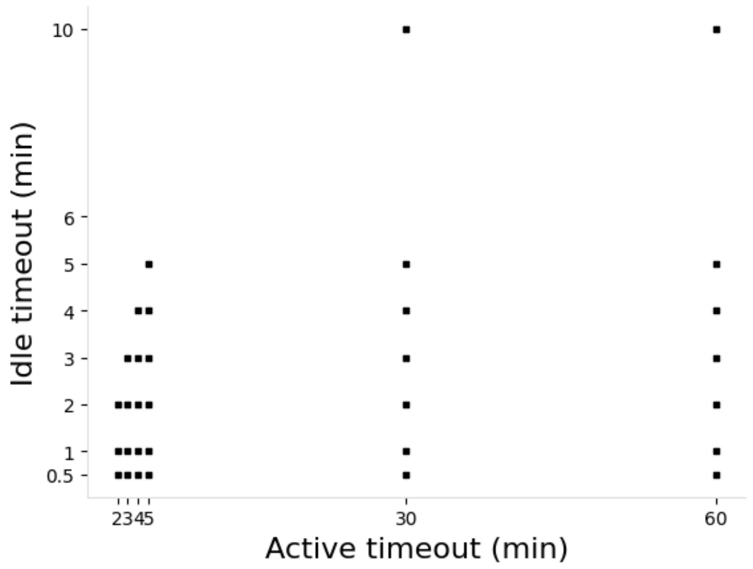


Figure 6.3: All 32 combinations of idle and active timeouts

To thoroughly evaluate the proposed methodology, we consider 32 combinations of idle and active timeouts. Figure 6.3 presents all the combinations used for evaluation, the idle timeout (min) take values from [0.5, 1, 2, 3, 4, 5, 6, 10] and the active timeouts (min) varies within the list [2, 3, 4, 5, 30, 60]. It is important to notice that the idle timeout is always less than or equal to the active timeout (idle $\leqslant$ active). For each tuple, we train ETC, RFC, and MLP models and we measure the performance in terms of four well-established metrics commonly used for intrusion detection: F1-score, Accuracy, Recall, and Precision. We record the best and worst timeouts for each dataset and model. The 'best timeout' denotes the specific combination of idle and active timeouts used for feature extraction, which yielded the highest performance for the ML model. Conversely, the 'worst timeout' represents the combination that resulted in the poorest model performance.

Experiments presented in this study were provided by the computing facilities of High Performance Computing simlab-cluster, of Mohammed VI Polytechnic University at Benguerir.

6.4.2 Performance evaluation

In the following, we present all the experiments and evaluations performed to study the impact of idle and active timeouts on the performance of NIDS ML models. In the first set of experiments, we measure the network intrusion detection performance of ETC, RFC, and MLP models for each datasets using the NFStream standard feature set computed based on the 32 pairs of idle and active timeouts. Tables 6.2, 6.3, 6.4, and 6.5 present the recorded best and worst idle and active timeouts (T_a, T_i) tuples and their performance difference for USTC-TFC16, CIC-IDS2017, UNSW-NB15, and CUPID datasets, respectively. For each dataset, we observe a unique (T_a, T_i) tuple that achieves the optimal performance, showing that the best timeout for a dataset can be the worst in another dataset. For instance, the pair *(1, 60)* is the best timeout tuple when the CIC-IDS2017 dataset is used, however, it is the worst in the case of UNSW-NB15 dataset. Furthermore, even if we focus on one dataset it becomes evident that there is no timeout tuple that excels with all three ML models. A timeout tuple could, without varying the environment (dataset), score the best results with one model and the worst results with another. For instance, in Table 6.5, for the same dataset CUPID, the timeout tuple *(10, 30)* achieves the best score when using the ETC model, but the worst with the RFC model. It is worth mentioning that when using the CIC-IDS2017 dataset the best performances are scored when the active timeout is large (60 minutes) and the idle timeout is small (1 minute and 0.5 minutes). When analyzing the F1-score performance difference between the best and worst timeouts, we note two major points: 1) ETC and RFC models are more stable than MLP. In tables 6.2, 6.3, 6.4, and 6.5 the computed performance difference of the MLP model is always greater than 6% and can reach up to $\sim$14%, however, the computed performance difference of the ETC and RTC models are less than 2.5% except for the case of CIC-IDS2017 dataset. 2) For some datasets (CIC-IDS2017), the performance difference shows that it is mandatory to fine-tune the idle and active timeouts. Yet, for some datasets (CUPID, UNSW-NB15) and with the right model the idle and active timeouts have little impact on the performance.

Table 6.2: Performance evaluation of ML models using the NFStream standard feature set with USTC-TFC2016.

	Best timeouts			Worst timeouts			Performance Difference		
Metrics	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
F1-score (%)	87.19	87.35	82.42	85.91	84.92	76.30	1.28	2.43	**6.12**
Recall (%)	86.63	86.44	82.84	84.84	85.66	77.40	1.79	0.78	5,44
Precision (%)	88.68	89.55	87.47	88.83	85.74	80.19	-0.15	3.81	7.28
Accuracy (%)	90.50	90.44	87.57	89.25	89.95	83.97	1.25	0.49	3.6
Timeouts	(10, 60)	(2, 2)	(2, 3)	(0.5, 3)	(4, 5)	(4, 60)	-	-	-

Table 6.3: Performance evaluation of ML models using the NFStream standard feature set with CIC-IDS2017.

	Best timeouts			Worst timeouts			Performance Difference		
Metrics	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
F1-score (%)	94.37	92.10	92.70	88.08	84.11	78.51	6.29	7.99	**14.19**
Recall (%)	91.02	88.77	90.75	85.41	82.47	81.37	5.61	6.3	9.38
Precision (%)	99.75	99.72	97.39	93.17	86.45	83.75	6.58	13.27	13.64
Accuracy (%)	99.78	99.79	99.76	99.77	99.76	99.75	0.01	0.03	0.01
Timeouts	(1, 60)	(0.5, 60)	(0.5, 60)	(3, 5)	(3, 5)	(2, 4)	-	-	-

Table 6.4: Performance evaluation of ML models using the NFStream standard feature set with UNSW-NB15.

	Best timeouts			Worst timeouts			Performance Difference		
Metrics	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
F1-score (%)	73.95	73.32	56.13	72.98	72.64	49.16	0.97	0.68	**6.97**
Recall (%)	70.11	69.40	53.76	69.29	68.59	47.87	0.82	0.81	5.89
Precision (%)	83.32	83.18	64.42	81.07	79.67	71.43	2.25	3.51	-7.01
Accuracy (%)	99.07	99.08	98.91	99.05	99.08	98.80	0.02	0	0.11
Timeouts	(3, 4)	(1, 30)	(0.5, 4)	(0.5, 60)	(2, 60)	(1, 60)	-	-	-

Table 6.5: Performance evaluation of ML models using the NFStream standard feature set with CUPID.

	Best timeouts			Worst timeouts			Performance Difference		
Metrics	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
F1-score (%)	93.74	92.77	91.81	92.53	92.11	84.68	1.21	0.66	**7.13**
Recall (%)	91.23	90.25	89.25	90.03	89.61	80.18	1.2	0.64	9.07
Precision (%)	99.19	99.18	98.2	99.13	99.06	96.42	0.06	0.12	1.78
Accuracy (%)	96.29	96.18	96.60	95.95	95.60	96.21	0.34	0.58	0.39
Timeouts	(10, 30)	(2, 30)	(1, 3)	(4, 30)	(10, 30)	(3, 30)	-	-	-

In our forthcoming series of experiments, we redirect our focus toward examining the relationship between flow features and both idle and active timeouts. We used all flow features described in Section 6.3.1 which consists of the features proposed by NFStream [12], the features selected in the work [33], the features introduced in the open source framework [130], and features developed by our team (see Table 6.1). Similar to the previous set of experiments, tables 6.6, 6.7, 6.8, and 6.9 present the best and worst obtained results with their difference when all flow features are used with all 32 idle and active timeouts combinations. The MLP model performance has significantly increased, for instance, the F1-score increased from 82.42% to 89.75% with the USTC-TFC2016 dataset. Similarly, the performance is on par or better when ETC or RFC are used with the expanded flow feature set. Furthermore, we observe that the ETC model's likelihood to achieve higher performance is big when the active timeout is large (60 minutes). However, considering all the scores there is no timeout pair that achieves top performance with all datasets and models.

Table 6.6: Performance evaluation of ML models using all flow features (from 4 sources) with USTC-TFC2016.

	Best timeouts			Worst timeouts			Performance Difference		
Metrics	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
F1-score (%)	91.50	91.62	89.75	90.47	90.56	88.35	1.03	1.06	**1.4**
Recall (%)	91.06	90.92	89.30	89.72	89.75	88.11	1.34	1.17	1.19
Precision (%)	92.89	93.61	92.39	93.10	93.13	91.51	-0.21	0.48	0.88
Accuracy (%)	94.68	94.77	93.89	93.99	94.04	93.27	0.69	0.73	0.62
Timeouts	(10, 60)	(2, 2)	(2, 3)	(0.5, 3)	(0.5, 2)	(3, 4)	-	-	-

Table 6.7: Performance evaluation of ML models using all flow features (from 4 sources) with CIC-IDS2017.

	Best timeouts			Worst timeouts			Performance Difference		
Metrics	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
F1-score (%)	94.15	92.50	94.19	88.13	84.80	77.13	6.02	7.7	**17.06**
Recall (%)	90.76	88.37	92.82	85.45	81.64	76.21	5.31	6.73	16.61
Precision (%)	99.93	99.85	99.07	93.22	93.14	78.70	6.71	6.71	20.37
Accuracy (%)	99.63	99.04	99.81	99.63	98.73	99.76	0	0.31	0.05
Timeouts	(10, 60)	(0.5, 60)	(0.5, 60)	(3, 5)	(3, 4)	(3, 4)	-	-	-

Table 6.8: Performance evaluation of ML models using all flow features (from 4 sources) with UNSW-NB15.

	Best timeouts			Worst timeouts			Performance Difference		
Metrics	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
F1-score (%)	74.63	73.87	63.30	73.55	73.06	60.61	1.08	0.81	**2.69**
Recall (%)	70.64	69.58	59.44	69.11	68.54	55.77	1.53	1.04	3.67
Precision (%)	83.29	84.34	71.34	82.60	90.60	76.19	0.69	-6.26	-4.85
Accuracy (%)	99.11	99.10	98.99	99.09	99.08	98.92	0.02	0.02	0.07
Timeouts	(5, 5)	(5, 60)	(1, 4)	(0.5, 30)	(1, 2)	(2, 2)	-	-	-

Table 6.9: Performance evaluation of ML models using all flow features (from 4 sources) with CUPID.

	Best timeouts			Worst timeouts			Performance Difference		
Metrics	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
F1-score (%)	95.33	94.91	94.57	94.22	92.79	89.68	1.11	2.12	**4.89**
Recall (%)	93.01	92.51	92.15	91.73	90.23	91.92	1.28	2.28	0.23
Precision (%)	99.43	99.39	99.30	99.36	99.24	92.01	0.07	0.15	7.29
Accuracy (%)	97.26	97.11	96.81	96.86	96.36	96.92	0.4	0.75	-0.11
Timeouts	(2, 60)	(5, 5)	(4, 30)	(2, 5)	(3, 3)	(4, 4)	-	-	-

Since the scores increased when all flow features were used, we performed various experiments to ascertain the relative importance of features and to identify any relationship between features and timeout pairs. Thus, we build two new feature sets using a feature selection strategy that consists of 1) computing features' importance using the ETC algorithm; 2) sorting features based on their importance; and 3) selecting 30% and 50% from the sorted list of features. Next, we evaluate the ETC model with the two new feature sets and all combinations of the active and idle timeouts. We justify the selection of the ETC model to perform these evaluations by the fact that, overall, it consistently exhibits higher levels of effectiveness and stability in the previous set of experiments. Tables 6.10, 6.11, 6.12, and 6.13 presents the obtained best and worst measures when using the ETC model with feature selection strategy. We noticed that the likelihood of scoring the best performances is high when both timeouts are large (idle $\geqslant$ 5 minutes, active $\geqslant$ 30 minutes). However, the application of the feature selection strategy has not yielded substantial improvements in model performance when compared with evaluations with all flow features. In fact, there are instances where implementing feature selection has resulted in a decline in the model's performance in terms of different metrics.

Table 6.10: ETC model performance evaluation using feature selection with USTC-TFC2016.

	Best timeouts		Worst timeouts	
Metrics	ETC (30%)	ETC (50%)	ETC (30%)	ETC (50%)
F1-score (%)	91.36	**91.53**	90.51	90.51
Recall (%)	90.96	91.06	89.72	89.73
Precision (%)	92.91	92.94	93.08	93.13
Accuracy (%)	94.62	94.69	94.02	94.02
Timeouts	(10, 30)	**(10, 60)**	(0.5, 30)	(0.5, 30)

Table 6.11: ETC model performance evaluation using feature selection with CIC-IDS2017.

	Best timeouts		Worst timeouts	
Metrics	ETC (30%)	ETC (50%)	ETC (30%)	ETC (50%)
F1-score (%)	**93.44**	93.43	87.24	87.74
Recall (%)	89.5	89.47	83.47	84.17
Precision (%)	99.90	99.91	93.19	93.28
Accuracy (%)	99.63	99.64	98.89	99.59
Timeouts	**(10, 60)**	(10, 30)	(2, 5)	(4, 5)

Given all the experiments done so far, we can answer the two research questions (**RQ1** and **RQ2**) related to the relationship between idle and active timeouts with flow features and models performances. Overall, we notice that the ETC model achieves

Table 6.12: ETC model performance evaluation using feature selection with UNSW-NB15.

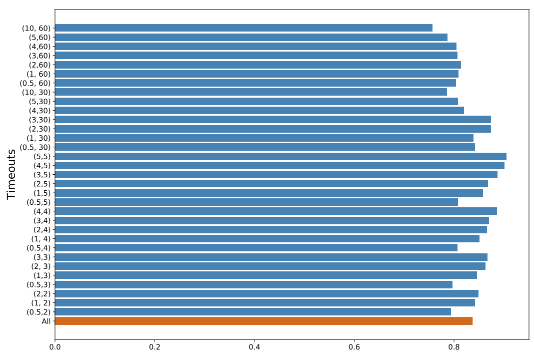
	Best timeouts		Worst timeouts	
Metrics	ETC (30%)	ETC (50%)	ETC (30%)	ETC (50%)
F1-score (%)	**75.72**	75.01	74.97	74.30
Recall (%)	72.25	71.24	71.57	70.57
Precision (%)	85.84	87.31	82.79	83.64
Accuracy (%)	99.10	99.11	99.09	99.10
Timeouts	**(2, 30)**	(4, 30)	(0.5, 3)	(1, 30)

Table 6.13: ETC model performance evaluation using feature selection with CUPID.

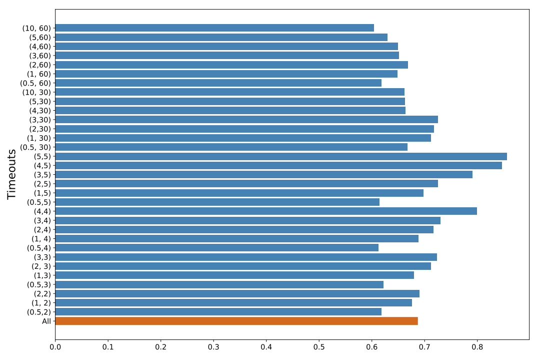
	Best timeouts		Worst timeouts	
Metrics	ETC (30%)	ETC (50%)	ETC (30%)	ETC (50%)
F1-score (%)	**95.29**	95.16	94.64	94.56
Recall (%)	92.95	92.81	92.18	92.02
Precision (%)	99.42	99.42	99.33	99.47
Accuracy (%)	97.28	97.18	97.22	97.6
Timeouts	**(5, 5)**	(2, 60)	(1, 4)	(0.5, 2)

superior performance when using all the flow features with larger idle and active timeouts. This is explained by the fact that larger timeouts gather more information about a distinct flow which makes intrusion detection more accurate. Nevertheless, this advantage is counterbalanced by the increased resource consumption, particularly in terms of larger flow caches.

In the next set of experiments, we explore a real-case scenario of distributed NIDS to answer the third research question **RQ3**. *FedAvg* algorithm was used to train a global model across 32 clients, each owning data extracted using a unique (T_a, T_i) combination. Figures 6.4, 6.5, 6.6, and 6.7 provide a visual representation of accuracy and F1-score when evaluating the MLP classifier on the test set of each client as well as all test sets combined for USTC-TFC2016, CICIDS2017, UNSW-NB15, and CUPID datasets respectively. Notably, the model exhibits varying performance levels across different clients for the USTC-TFC2016 dataset. In other words, the performance of the global model significantly fluctuates from client to client as idle and active timeouts change, showing a difference of up to 25% in the F1-score. However, while the global model may not demonstrate any noticeable performance improvement on the entire test set of USTC-TFC2016, the results obtained from CIC-IDS2017, UNSW-NB15, and CUPID datasets suggest a significant potential for enhancement. The global model exhibits the capacity to achieve a performance level that, although falling short of being the absolute best, remains commendably satisfactory to a certain degree. These findings emphasize that the application of the *FedAvg* algorithm led to

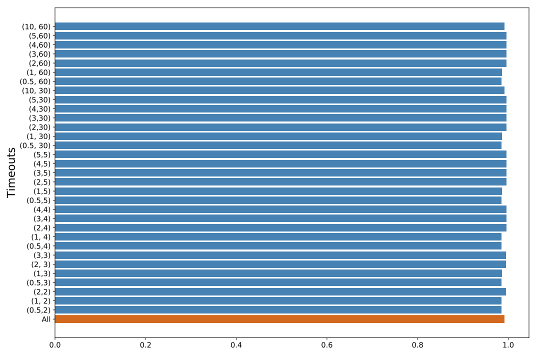


(a) Accuracy

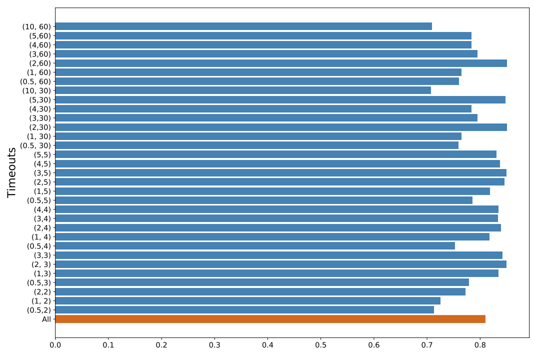


(b) F1-score

Figure 6.4: Performance evaluation of FL for distributed NIDS with different timeouts on USTC-TFC2016 dataset.



(a) Accuracy

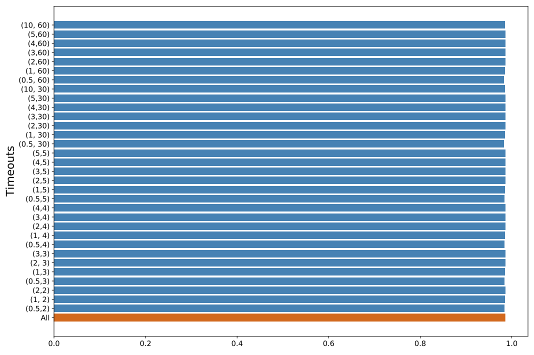


(b) F1-score

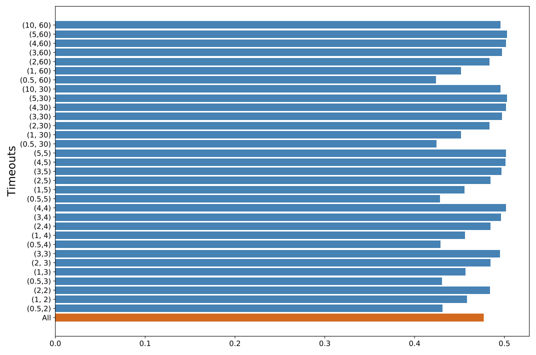
Figure 6.5: Performance evaluation of FL for distributed NIDS with different timeouts on CIC-IDS2017 dataset.

fairly good performance enhancement within distributed NIDS with different timeouts. Consequently, this outcome underscores the potential of FL and the need for further in-depth investigations and studies of FL algorithms that consider different NIDS with distinct characteristics resulting from different timeout configurations.

On a more holistic level, we conducted one last analysis leveraging results from previous experiments. The main idea is to evaluate whether a specific timeout tuple achieves better performance in comparison to the others. For each timeout tuple, we performed a total number of 36 experiments: 1) the first set of experiments contains 12 measures for each timeout tuple (4 datasets and 3 models, see Tables 6.2, 6.3, 6.4, and 6.5); 2) the second set of experiments consists of two main sets of experiments, first 12 experiments for each timeout (4 datasets and 3 models, see Tables 6.2, 6.3, 6.4, and 6.5) then 8 experiments which use one model with two different flow features (4 datasets and 2 flow feature sets, see tables 6.10, 6.11, 6.12, and 6.13); 3) The last evaluations performed 4 experiments for each timeout tuple (4 datasets, see Figure

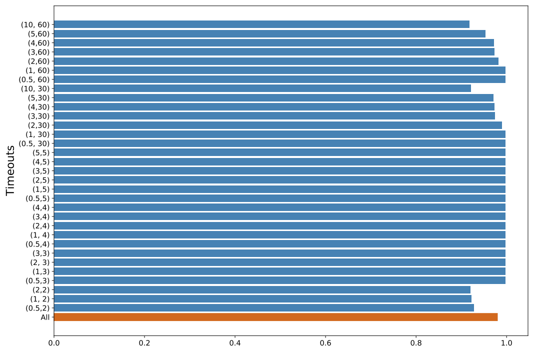


(a) Accuracy

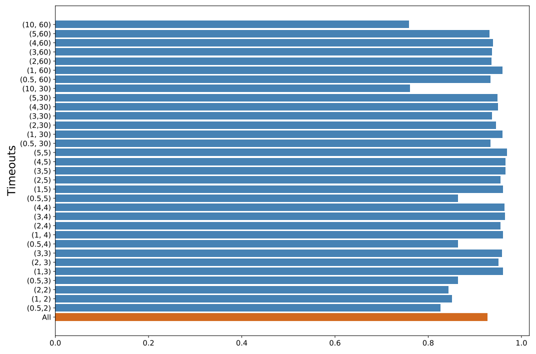


(b) F1-score

Figure 6.6: Performance evaluation of FL for distributed NIDS with different timeouts on UNSW-NB15 dataset.



(a) Accuracy



(b) F1-score

Figure 6.7: Performance evaluation of FL for distributed NIDS with different timeouts on CUPID dataset.

6.4). This analysis is visually represented in the form of a boxplot illustrating the F1-scores for various (T_a, T_i) combinations, as depicted in Figure 6.8. The latter shows a minimal variance in model performance across the diverse timeout combinations and no specific timeout setting appears to stand out as the clear choice for an initial starting point. This finding underscores the complexity of selecting an ideal timeout combination right from the outset.

In light of these results, it is evident that the choice of timeout can influence NIDS ML models' performance by as much as 17% (Table 6.7). On the one hand, the ETC model is stable and scores better results when both the idle and active timeouts were set to larger values. On the other hand, the MLP model is the most impacted by timeout selection, this is important to notice given the current deep learning landscape. Figure 6.8 provides an answer to **RQ4** and shows that no one-size-fits-all timeout tuple exists, and the choice depends on the environment of the application (dataset) and the ML model. Consequently, incorporating a fine-tuning stage into the model

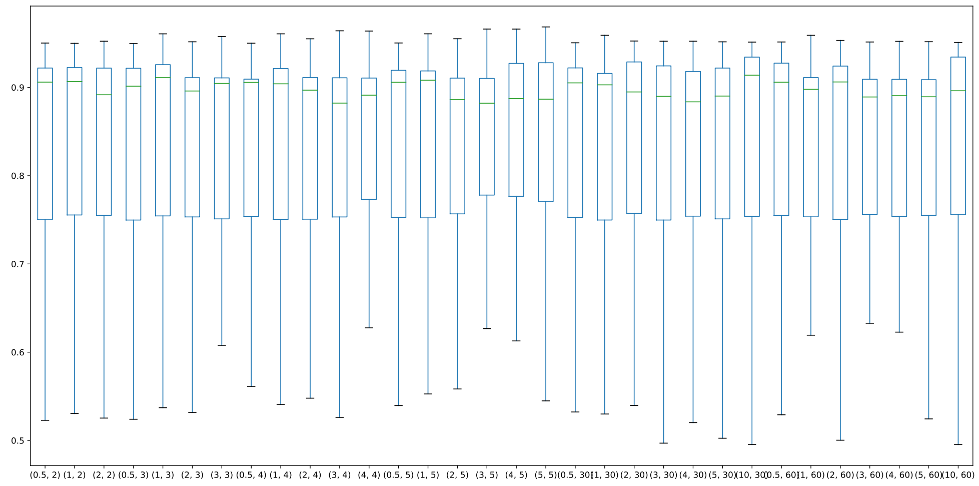


Figure 6.8: Boxplot depicting F1-score variation of different timeouts considering all preceding experiments.

development process mitigates the risk of settling for an inefficient timeout thereby enhancing the overall effectiveness of the ML-based network intrusion detection.

6.5 Summary

The aim of this chapter was to shed light on the relationship between flow timeout and the performance of ML models. Through extensive experimentation, we investigated the impact of varying the values of idle and active timeouts on the quality of the extracted features and therefore on ML models used in the context of NIDS. We utilized 32 combinations of idle and active timeouts to build 32 versions of each NIDS dataset deployed in this study, namely, USTC-TFC2016, CIC-IDS2017, UNSW-NB15, and CUPID. For each version, we train three ML models, ET, RF, and MLP classifiers. Our findings demonstrate that adopting a one-size-fits-all approach to timeouts can lead to suboptimal performance of ML models. Specifically, we found that different combinations of idle and active timeouts had a substantial impact on the accuracy and performance of the NIDS models, with a significant difference in performance between the most effective timeout combination and the least effective. Furthermore, the results indicated that the ETC and RFC models exhibit greater stability and resilience to fluctuations in idle and active timeouts compared to the MLP model. Notably, ET classifier also exhibited superior performance when used with longer timeouts. The study also revealed that extending the feature set reduces the variance in models' performance considering different timeouts. Through the application of FL, we also demonstrate its promising potential in handling distributed NIDS with diverse timeout configurations.

Our analysis has provided valuable insights into the impact of flow timeout on the performance of NIDS, underscoring the need for network security professionals and data scientists to pay meticulous attention to the selection and configuration of idle and active timeouts when training ML models for NIDS.

7 Federated Learning for Heterogeneous Systems

7.1 Introduction

FL has shown promise as an effective approach for improving the accuracy and efficiency of IDS in network security. However, FL is distinguished from traditional distributed optimization methods by several key challenges. FL faces hurdles in numerous applications, particularly in intrusion detection.

Communication overhead. A key limitation inherent in any FL application arises from the associated communication costs during each training round [132]. FL operates by exchanging model updates, first from the server to the selected clients and then from each client back to the server. With more devices, large models, and constrained networks, the communication overhead increases significantly demanding substantial bandwidth. Moreover, in scenarios where the participating devices have diverse computational capacities, the synchronization of model updates becomes more intricate. These challenges intensify with larger and more dynamic FL environments. This issue has been addressed in the literature through the use of compression techniques (such as quantization and sparsification) and the reduction of the communication rounds [66]. However, this generally leads to a trade-off between communication efficiency and model accuracy, as reducing the communication overhead may result in loss of information, consequently compromising the effectiveness of the intrusion detection model. In the context of intrusion detection, real-time analysis of network data is crucial, as well as fast and accurate detection. Hence, the efficiency and speed of communication become crucial. This becomes more challenging in large-scale networks with millions of connected devices. In such a case, even if only 1% of clients are selected at each round, the communication of at least 10,000 updates is performed, placing a heavy burden on the FL infrastructure and potentially causing delays in model convergence.

Federated poisoning attacks. FL's strength lies in distributed data, keeping information secure and private. However, this decentralization shifts the target, creating new security challenges for the devices that store and process this data locally, which can become targets for malicious actors. Attackers inject manipulated data into the training process, for instance by modifying the labels (label-flipping attack), consequently biasing the global model and generating poisoned predictions. Another type of poisoning attacks directly targets the model. Attackers tamper with the global model, introducing subtle but harmful perturbations. Poisoned models can generate a wave of false alarms and misclassify legitimate threats or overlook real attacks, leaving the network vulnerable [133]. One way to tackle this issue is to rely on network management to guarantee that only devices adhering to the intended behavior are permitted to take part in the training process [134]. However, these methods may impose additional communication overhead. In their paper, Nguyen et al. [133] proposed a poisoning attack for FL-based IDS, by manipulating the traffic data, showcasing how subtle manipulations can create significant damage. Hence, the need for robust defense mechanisms against federated poisoning attacks while preserving the effectiveness of IDS is crucial.

Privacy concerns. FL emerged as a solution primarily aimed at alleviating the privacy concerns associated with centralized ML models. However, despite these privacy-centric intentions, it's crucial to recognize that the privacy of participants' data in FL is not completely guaranteed [135]. Several challenges contribute to this uncertainty. The model weights encapsulate the features derived from each device's data, and may still contain sensitive information or patterns from the local data. The process of exchanging these weights between the central server and individual devices poses potential vulnerabilities. Attackers might analyze and decode the weights to reconstruct sensitive features, possibly causing harm if misused. Furthermore, the aggregation of model updates at the server introduces inherent risks. The server gains access to information derived from the data on individual devices. If not properly secured, this aggregated information could become a target for a malicious server or adversaries seeking to exploit vulnerabilities. Privacy-preserving techniques including Differential Privacy (DP) [136], Secure-Multiparty Computation (SMC) [137], and Homomorphic Encryption (HE) [138] are utilized to secure data submission. Recent efforts are exploring the application of these techniques for IDS. For instance, Li et al. [139] proposed the use of HE to secure the model's updates exchange with the server in a FL-based intrusion detection framework for Industrial Cyber-Physical Systems. Another work [140] has explored the implementation of DP within an FL-based IDS, with a specific focus on handling non-IID data. These methods, however, often sacrifice accuracy and efficiency, potentially hurting the ability to detect attacks in IDS. Therefore, further research is required to tackle these challenges and uphold the initial promise of enhancing privacy while leveraging the power of collaborative learning.

Data heterogeneity. Data heterogeneity, also known as non-IID data, is one of the key challenges in FL [141], confining its adoption in real-world applications. Each client in an FL setting might have data drawn from different domains, subject to specific biases, or influenced by unique user behaviors. This mismatch in data distributions leads to clients learning conflicting or irrelevant patterns, hindering the overall model's generalization. This is particularly evident in the context of IDS, where the properties and characteristics of packets vary greatly based on many factors such as time and usage. These variations in data characteristics across different clients can lead to challenges in model training, aggregation, and generalization. Models trained on non-IID data may struggle to generalize well across all clients, leading to issues such as poor convergence, biased model parameters, and suboptimal predictive performance. Dealing with non-IID data in FL requires specialized techniques and algorithms that can adapt to the diversity and heterogeneity inherent in decentralized data sources. To this end, numerous techniques have been proposed in the literature which have been surveyed in [142]. In the next section, we address the challenge of non-IID data and its impact on the performance of DL models, in the specific case when trained with BN layers.

7.2 Problem statement

Generally, non-IID data have not been an obstacle towards learning from distributed data since data centers have access to the entire dataset and can distribute the data according to IID assumption. However, in FL, the server does not have such a luxury, as it does not have access to local client data. In real-world scenarios, each client may correspond to an end device, a particular user, or a geographic location. Thus, the data available locally are more likely to be dependent and non-identical. Each partition has a different probability distribution that fails to represent the population distribution. For example, data collected from mobile devices will reflect the preferences of each user based on his age, location, gender, etc. Another example is the deployment of IDS in a federated setting with heterogeneous systems that include, for instance, health care centers, vehicular networks, smart cities, and cyber-physical systems altogether. This may introduce unique challenges and opportunities due to the diverse nature of the components and technologies involved. Heterogeneous systems typically consist of a mix of different hardware architectures, operating systems, and network protocols, making them more resilient but also more complex. Integrating IDS into such environments makes it challenging to learn a global model that performs well on all client partitions. The difference in local distributions makes learning from local data a challenging task due to the inconsistency between 1) the results of the gradient algorithm executed locally on each client k that aims to minimize a local objective function $F_k(w)$ over m_k samples and 2) the global objective of minimizing the global function $F(w)$ over all data samples $\sum_k m_k$.

It is known that BN becomes ineffective in certain settings (small and non-IID mini-batches [143]). On the one hand, BN relies on statistics computed from the mini-batch, such as mean and variance, to normalize the inputs. In small mini-batches, these statistics may not accurately represent the true distribution of the data, leading to suboptimal normalization. When the mini-batch size is small, the computed mean and variance may be noisy estimates of the true population statistics. This noise can introduce variability in the normalization process, hindering BN's effectiveness. In extreme cases, small mini-batches might result in unstable training. On the other hand, BN assumes that the samples in a mini-batch are drawn from the same distribution. In scenarios where the mini-batch contains diverse or non-representative samples, BN may not effectively normalize the inputs. Non-IID mini-batches violate the assumption of homogeneous data distribution within the batch. For example, if a mini-batch contains samples from different classes or domains, the computed statistics may not accurately capture the characteristics of any specific subset, leading to normalization issues. In fact, if μ_B and σ_B diverge from the global mean and variance of the whole dataset, then the testing accuracy will be unsurprisingly low. In FL settings, where each data partition could differ from the other, the performance degradation is more severe as μ_B and σ_B vary significantly across partitions.

7.3 Preliminaries

BN has become an indispensable technique in deep learning and it is one of the components that has helped boosting ML applications in the last few years. In order to stabilize and speed up neural network training, BN normalizes the input distribution to have a mean of zero and a variance of one. BN acts differently between the training and inference phases. During training, BN uses mini-batch means and variances for normalization. In fact, for each mini-batch B, BN first computes the mean μ_B and the variance σ_B, and then normalizes each input x_i in B as follows:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

where ϵ is a small constant added for numerical stability. Further, BN adds two learnable parameters, namely the γ coefficient and the β bias, to automatically scale and shift the normalized pre-activations. In other words, these parameters allow the network to convert the mean and variance to their optimal values:

$$y_i = \gamma \hat{x}_i + \beta$$

At the inference time, as each sample is processed separately, computing the mini-batch mean and variance is infeasible. Thus, during the training, we estimate the mean and variance over the whole dataset as follows:

$$E = \frac{1}{m} \sum_{i=1}^{j} \mu_B^{(i)} \qquad Var = (\frac{m}{m-1}) \frac{1}{m} \sum_{i=1}^{j} \sigma_B^{2^{(i)}}$$

where m is the mini-batch size and j is the total number of mini-batches. These estimated values are then used at the inference time as follows:

$$y = \frac{\gamma}{\sqrt{Var + \epsilon}} x + (\beta + \frac{\gamma E}{\sqrt{Var + \epsilon}})$$

7.4 Methodology

In the case of FL with IID settings, training neural networks with BN layers does not pose any problem since each local data is drawn from the global distribution, and therefore the mini-batch statistics (μ and σ^2) are representative of the whole dataset statistics. However, in non-IID setting, the statistics calculated by each client are biased to its local data and vary across clients.

In spite of previously discussed alternatives showing some promising results which helped maintain the performance of BN in a few cases [144], they come with some side effects. For instance, when using GN, one should be careful with the batch size as the performance of GN degrades with larger batch sizes. Furthermore, Fixup Resnet is limited to Resnets and also requires a carefully-tuned learning rate strategy. Thus, the aforementioned conclusions are more nuanced than previously thought ,hence, more thorough studies and experimentation are needed before concluding if any of these methods can replace BN for different applications and CNN/DNN models.

To address the problem of BN in FL under non-IID data, we proposed *FedBS*, a new aggregation strategy that handles batch parameters in FL during training to improve the models' performance against heterogeneity. *FedBS* assumes that the global model used in FL has BN layers, and modifies the aggregation strategy at the server side.

In their paper , Y. J. Cho et. al [75] analyze the convergence of *FedAvg* under biased selection strategies and prove that biasing sampling procedure of participating clients towards those with higher loss helps with convergence time as well as higher accuracy. Based on the conclusions drawn from Y. J. Cho et. al analysis, we propose a modification to the naive approach of weighting local models in *FedAvg*. *FedAvg* uses an unbiased selection strategy for clients, where the clients are chosen randomly and their weights are computed based on their local data size. This strategy fails in cases from which we mention when a client has a large data drawn from one class and/or contains outliers. *FedBS* modifies *FedAvg* in two major steps. First, in the early rounds where $F_k(w)$ highly differs from a client to another, *FedBS* weighs the clients according to their respective loss during training. Specifically, the higher the loss of a local model the higher shall be the respective weight of the corresponding client in the aggregation step. This step helps to warm up the global model. When all clients start to converge to the global model and $F_k(w)$ is stable between clients we switch the training using equal weights for all clients and using *FedProx* algorithm. We check the stability of $F_k(w)$ by computing the standard deviation of the loss vector of the partaking clients, if $std(v) \leqslant \epsilon$ for r consecutive rounds, then we continue the training with *FedProx* algorithm and using equal weight for all clients ($\epsilon = 0.1$ and $r = 5$ for experimentation) (Algorithm 3).

Figure 7.1 provides a global overview for the first step of*FedBS* (without the weight exchange of the global model):

1. At each communication round, a subset k of clients is chosen randomly to participate in the current FL round. Each participating client after training on his

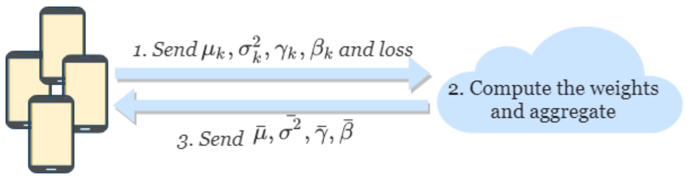

1. Send $\mu_k, \sigma_k^2, \gamma_k, \beta_k$ and loss
2. Compute the weights and aggregate
3. Send $\bar{\mu}, \bar{\sigma^2}, \bar{\gamma}, \bar{\beta}$

Figure 7.1: Overview of *FedBS*

local data, shares the BN parameters used in the normalization layers with the central server, as well as the loss value obtained in the current round.

2. The server, based on the local loss values, calculates the importance of each client, i.e. higher weight is given to the client with the highest local loss. Then aggregates all the local models.

3. The updated parameters are sent back to the newly selected clients for the next round. Each client updates both training and inference statistics based on those received from the server

7.5 Experiments and results

Datasets and non-IID settings We conduct our experiments on three datasets, Mnist, Fashion-Mnist and Cifar-10. For Cifar-10 [145] dataset, We use a CNN model with four convolution layers, a first 5x5 convolution layer followed by three 3x3 convolution layers, with 100, 150, 250, 500 channels respectively. Each convolution layer is followed by a normalization layer, a Relu activation, and a 2x2 max pooling. In the end, three fully connected layers were used (with 4500, 270, 150 units resp.). For Mnist [146] and Fashion-Mnist [147], we use a CNN model with two 5x5 convolution layers, the first one with 16 channels, the second one with 32. Each convolution is followed with a BN layer, ReLu activation, and a 2x2 max pooling, then a fully connected layer with 1568 units.

We consider two distributions, for non-IID Data partitions. The first is *balanced label distribution skew* [72] where the Mnist dataset is divided into 200 shards of 300 images. We consider 100 clients, each receives two shards, and thus has samples from only two classes. The same setting is used to partition Fashion-Mnist and Cifar-10. The second is *unbalanced label distribution skew*, where we divide the datasets into 1200

Algorithm 3 *FedBS*: The K clients are indexed by k, E is the number of local epochs, $\mathcal{B}$ is a set of mini-batches each of size m, η is the learning rate, and ϵ is a positive small number.

```
Server executes:
    initialize $w_0$
    for each round $t = 0, 1, \ldots$ do
        m ← max(C×K, 1)
        $S_t$ ← (select a random set of m clients)
        send $w_t$ to each client k ∈ $S_t$
        for each client k ∈ $S_t$ do
            $w_{k,t+1}, F_k(w) \leftarrow ClientUpdate(k, w_t)$
        end for
        $W_k = \frac{F_k(w)}{\sum_{k \in S_t} F_k(w)}$
        $w_{t+1} = \sum_{k \in S_t} W_k w^k_{t+1}$
        until $std(F(w)) < \epsilon$
        switch to FedProx
    end for

ClientUpdate($k, w$):
    for each $i$ from 1 to $E$ do
        for batch $b \in \mathcal{B}$ do
            $w \leftarrow w - \eta \nabla \mathcal{L}(w; b)$
            computer $F_k(w)$
        end for
    end for
    return $w$ and $F_k(w)$ to server
```

shards of 50 samples each. Each client receives at least one shard and a maximum of 30 shards.

Baselines. We compare the proposed method *FedBS* to two standard algorithms in FL, *FedAvg* and *FedProx*. We also include the recently proposed method *FedBN*. Since *FedBN* does not deal with our setting of testing where the testing clients do not have any data to calculate the mean and variance on, we handle this by taking the average of the mean and the variance learned in the previous round as was done with the trainable parameters in the proposed generalization of *FedBN*.

Training. For all datasets, we divide the samples on 100 clients either evenly or unequally. For each communication round, 10 clients are chosen randomly to participate in the FL round. During training, we set batch size B to 10, learning rate r to 0.01, local epochs E to 10, and global rounds to 250 for Balanced and unbalanced

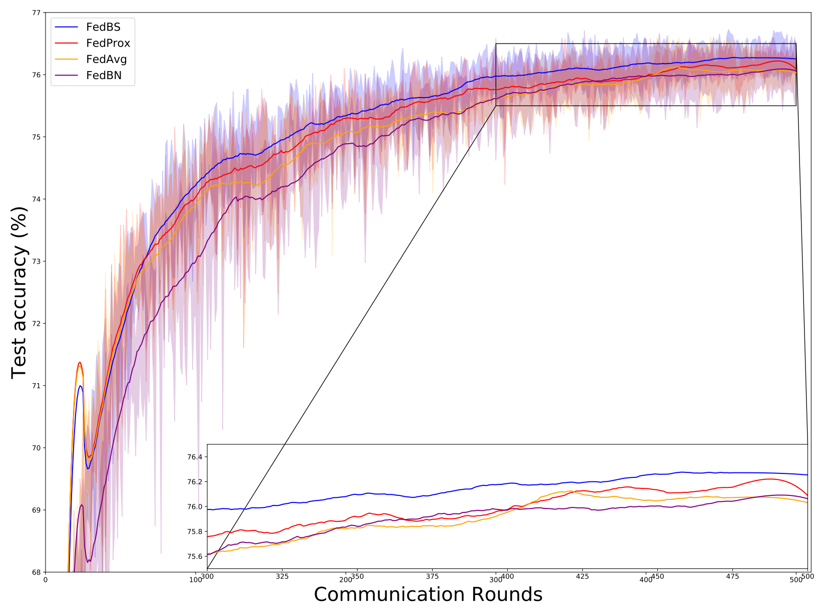


Figure 7.2: Testing results on Cifar-10 dataset partitioned unequally.

Mnist/Fashion-Mnist and 1000, 500 for Balanced and Unbalanced Cifar-10 respectively. We use SGD optimizer with cross-entropy loss function. For BN hyperparameters, we use the default values 0.1 for momentum and *ema* as the moving average technique.

Evaluation. We evaluate *FedBS* on the original test datasets as our general test sets, each containing 10000 samples. For each method, we run the training and test five times and we plot the mean result.

Discussion. Figure 7.2 and 7.3 provide the testing accuracy on Cifar-10 dataset in both cases equal and unequal partitioning of the data respectively. We observe that *FedBS* yields up to 2× faster convergence rate with higher test accuracy. We notice that *FedBN* is slower than all other methods including *FedAvg*. Moreover, compared to the baselines, *FedBS* goes smoother with less fluctuations for both cases. Similarly, we run tests on Mnist and Fashion-Mnist datasets (Figure 7.4 and 7.5). For both scenarios, equal and unequal partitioning of the data, *FedBS* is on par or better than other FL methods.

Testing on unbalanced Cifar-10 dataset locally. We also tested *FedBS* in the local case where clients have access to all BN parameters (Figure 7.6). Although *FedBN* surpasses the other methods, however, it goes much slower than all of them during the first 200 rounds, which supports the remarks previously made about the slowness of *FedBN*. On the other hand, *FedBS*, *FedAvg*, and *FedProx* all converge to the same accuracy, however, *FedBS* shows faster convergence during earlier rounds.

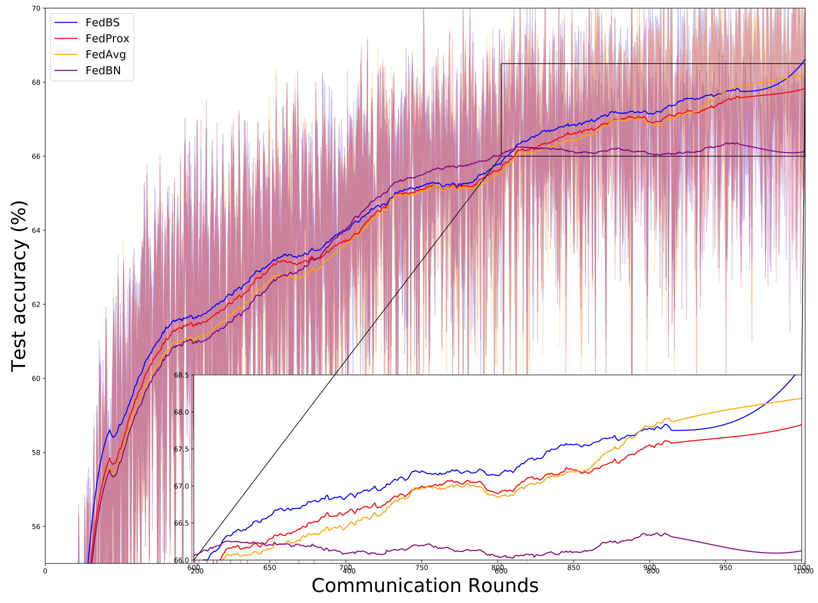


Figure 7.3: Testing results on Cifar-10 dataset partitioned equally.

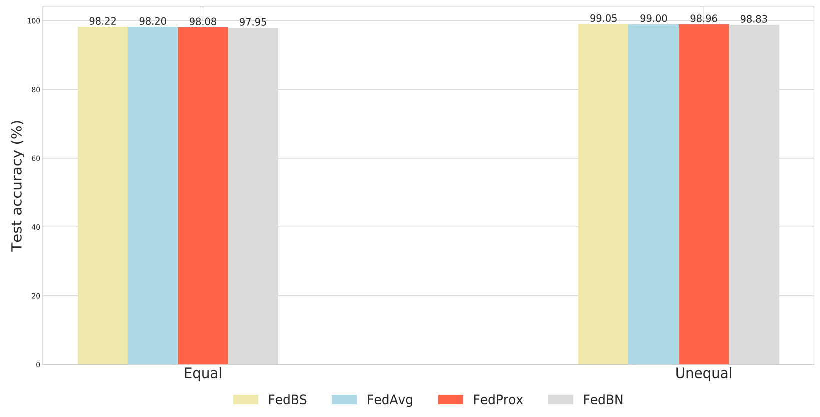


Figure 7.4: Testing results on balanced Mnist and Fashion-Mnist datasets.

7.6 Summary

This chapter introduced a novel approach, called *FedBS*, for handling DNN models with normalization layers in the FL setting. In order to deal with the non-IIDness assumption, *FedBS* warms up the global model by weighting the local models based

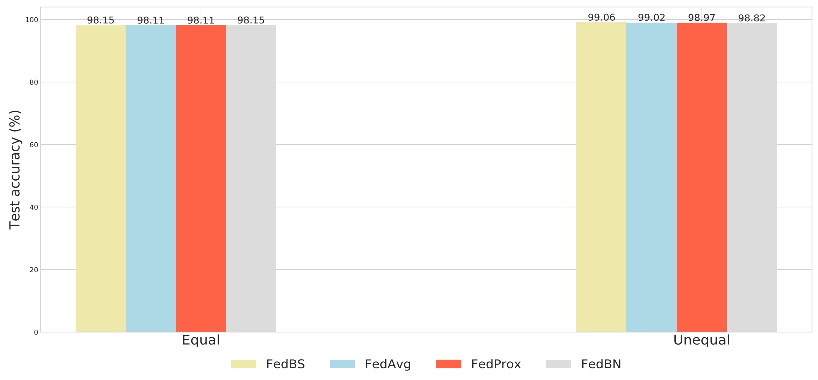


Figure 7.5: Experiments results on unbalanced Mnist and Fashion-Mnist datasets.

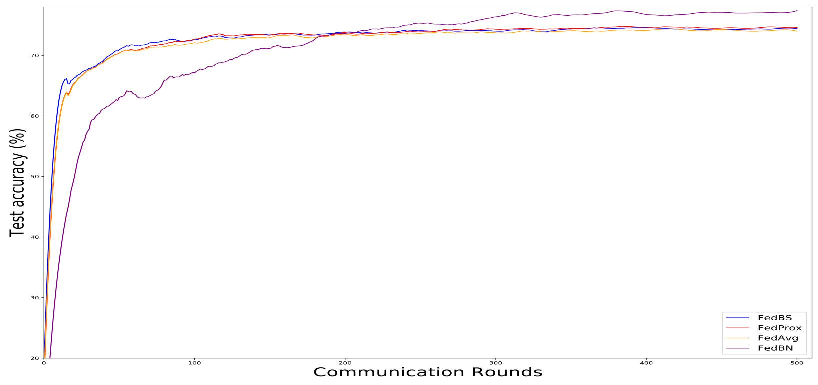


Figure 7.6: Testing results on Cifar-10 dataset partitioned unequally (Local test).

on each client's loss value, then *FedBS* switches to equal weighting with *FedProx* algorithm. We have comparatively evaluated *FedBS* against *FedAvg*, *FedProx* and *FedBN* using a multitude of datasets under various non-IID settings. First, based on exhaustive experimentation conducted on the known and complex dataset Cifar-10, *FedBS* outperformed all the state-of-the-art approaches in terms of both convergence rate and model performance with a scale up 2× faster. Furthermore, throughout all experiments run on Mnist and Fashion-Mnist datasets, *FedBS* was either on par or better than the concurrent methods. Finally, since batch statistics can be used by potential attackers, and as future directions we intend to propose a more secure communication of batch parameters between the clients and the server.

8 Conclusion and Future Work

In this concluding chapter, we summarize our contributions to IDS and FL fields, discuss the major findings of this thesis, highlight the inherent limitations of the proposed approaches, and conclude by providing an outlook on potential future research directions.

8.1 Summary

The concept of IDS has been around for more than four decades. However, it has only recently exploded onto the scene, becoming a cornerstone of modern network security. This is due to the increasing number and sophistication of attacks on computer networks. The thing that made firewalls and antiviruses insufficient to secure networks. The inadequacy of these passive measures has paved the way for the elevated role of IDS, which have now become an indispensable component for organizations, working side by side with firewalls and antiviruses, filling the gap left by them. In this thesis, we have identified and addressed three research problems in network intrusion detection: (i) the concerns associated with centralized approaches for NIDS, (ii) the relationship between the quality of network features and the performance of ML and DL models, and (iii) the problem of non-IID data in heterogeneous systems.

An efficient IDS should provide high detection accuracy of known and unknown threats, timely alerts, and minimal false alarms. Many works have focused on improving the performance of IDS through AI techniques, such as ML/DL algorithms and anomaly detection methods. However, most of these approaches rely on centralized data collection and processing, often overlooking the inherent challenges and risks such as data privacy. The first contribution of this dissertation tackled this challenge. We proposed *Fed-ANIDS*, an innovative approach that combines FL and autoencoders for intrusion detection in distributed networks. It addresses the need for both privacy preservation and robust intrusion detection. The autoencoders learn representations of normal data behavior while FL ensures privacy by training models locally on client subsets. Leveraging both FL and autoencoders allows for efficient training on diverse and potentially high-dimensional network data while preserving privacy. At the detection phase, any deviation from normal behavior is marked as an attack. *Fed-ANIDS* demonstrated its effectiveness on three publicly available datasets (USTC-TFC2016, CIC-IDS2017, and CSE-CIC-IDS2018), achieving high accuracy, low false alarms, and outperforming GAN-based models. Moreover, The chosen FL algorithm *FedProx*, consistently matches or surpasses the popular *FedAvg* in terms of performance. These results highlight the potential of using autoencoders for large-scale intrusion detection in distributed systems and pave the way to explore other FL algorithms.

The performance of ML/DL models is greatly influenced by the datasets they are trained on. However, researchers often concentrate on enhancing model performance, neglecting the critical role played by the quality of the training datasets and its impact on the overall performance. We studied the critical relationship between flow timeout settings, the quality of the extracted features, and the performance of ML/DL models used in NIDS. Through extensive experiments, we investigated how different idle and active timeout values affect the quality of extracted features and, consequently, the performance of ML/DL models for intrusion detection. The key findings are: 1) A one-size-fits-all approach to timeouts is ineffective. Different combinations of idle and active timeouts significantly impact the accuracy and performance of NIDS models. 2) ML models vary in their sensitivity to timeouts. Some models are more resilient to timeout fluctuations compared to others. 3) Feature engineering plays a key role in robustness. Expanding the feature set reduces performance variance and helps stabilize model performance across different timeout configurations. 4) FL holds promise for ditributed NIDS with varying timeouts. Applying FL demonstrated its potential to handle distributed NIDS with varying timeout settings.

FL allows training models across multiple devices without sharing raw data, but issues arise when data distributions across devices differ (non-IID setting). We introduced *FedBS*, a novel approach that tackles challenges associated with training DNNs with normalization layers in FL settings. Before averaging updates, *FedBS* weights local models based on their loss values, focusing on those performing well with their specific data. After the initial "warm-up", *FedBS* switches to the *FedProx* algorithm with equal weighting for all clients, ensuring stability and convergence. The authors compared *FedBS* against state-of-the-art FL methods like *FedAvg*, *FedProx*, and *FedBN* on various datasets and non-IID conditions. In most cases, *FedBS* excels, converging 2x faster and achieving superior model performance. In other cases, *FedBS* consistently matched or surpassed competing methods.

8.2 Research gaps and future work

Although we have made great strides in trying to solve the research problems stated above, we have been faced with other open problems:

- **Generalization**: Generalization is a common issue in FL. The generalization issue in FL refers to the difficulty of models trained on data from multiple devices/clients to perform well on unseen data, which can come from two scenarios: 1) New data within known domains: When the model encounters new data points from the same type of devices and environments it was trained on, but not specifically included in the training set. 2) Unseen domains: When the model faces data from entirely new devices, environments, or data distributions it has never seen before. This issue arises due to the inherent heterogeneity of data in FL settings, leading to reduced accuracy and biased decisions. We have evaluated how well *Fed-ANIDS* adapts to unseen data, highlighting the two previously mentioned scenarios: 1) New data from known domains: The autoencoder was trained on two datasets, leaving the third unseen. Even within these familiar domains, *Fed-ANIDS* performance dropped when tested on new data. 2) Completely unseen domains: The model was never exposed to the third dataset during training. As expected, performance suffered even more significantly in this scenario. Nevertheless, it is important to highlight that in certain instances, the model exhibited superior performance on an unseen dataset compared to a dataset used for training, and vice versa. Moreover, in another case, the model could not generalize either on the unseen dataset or on the one used for training but performed well on the second dataset used during training. We have explained these observations by the size of the datasets. The model is biased to the largest datasets whether they have been used for training or left unseen for evaluation.

- **Limitations of existing IDS datasets for FL** As previously discussed in 2.5, the majority of existing IDS datasets have major issues related to data collection, labeling, and feature extraction. These issues exacerbate in FL environments, making most of these datasets unsuitable for deployment. First, many IDS datasets lack information related to various IP addresses or devices, making it unfeasible to identify the entities within the FL system. Consequently, this hinders the ability to design a realistic FL scenario. Second, the curse of class imbalance reaches even IDS datasets [34]. Most of these exhibit a notable imbalance between benign and attack traffic, generally encompassing a restricted number and range of attacks. Various approaches have been proposed to alleviate the impact of the imbalance problem. One work [148] proposed the use of the Synthetic Minority Oversampling Technique (SMOTE) technique to handle the class imbalance issue in the CIC-IDS2018 dataset. Another paper [149] incorporated under-sampling and embedded feature selection in the data preprocessing phase to relieve the issue of class imbalance in the CIC-IDS2018 dataset. Despite these endeavors, there is a need for further research to build clean and realistic datasets specifically tailored for application in decentralized settings.

- **Intrusion detection on non-IID data** As demonstrated in the second contribution, network flows can be extracted using different parameters including flow timeout, resulting in distinct sets of features and datasets. Our empirical analysis has highlighted the significant influence of these parameters on the performance of ML/DL models. Consequently, in practical FL scenarios, individual devices or clients may possess datasets generated with varying parameters, thus producing datasets that deviate from the IID assumption. This observation underscores the importance of considering data heterogeneity in FL-based solutions for intrusion detection. Therefore, it is essential to develop FL algorithms and methodologies capable of accommodating and mitigating the impact of data heterogeneity in IDS, ensuring robust performance in real-world FL deployments.

For future work, we plan to address the aforementioned research gaps. Further research and evaluations of FL-based IDS models with a focus on domain generalization in order to improve their performance across different network domains and achieve better intrusion detection accuracy overall. Moreover, other FL algorithms should be explored and tested for this purpose. Furthermore,

Another direction is to further examine the impact of timeouts on different network security systems (NSM, Security information and event management (SIEM), etc.). In addition, we plan to explore heterogeneous FL algorithms to enhance the performance of ML/DL-based NIDS when different configurations of timeouts are used.

Bibliography

[1] Marshall A Kuypers, Thomas Maillart, and Elisabeth Paté-Cornell. "An empirical analysis of cyber security incidents at a large organization. Department of Management Science and Engineering". In: *Stanford University, School of Information, UC Berkeley, http://fsi. stanford. edu/sites/default/files/kuypersweis_v7. pdf, accessed July* 30 (2016), pp. 1–22.

[2] Eric Cole. *Network security bible*. John Wiley & Sons, 2011.

[3] Brian Caswell and Jay Beale. *Snort 2.1 intrusion detection*. Elsevier, 2004.

[4] Wonhyung Park and Seongjin Ahn. "Performance comparison and detection analysis in snort and suricata environment". In: *Wireless Personal Communications* 94.2 (2017), pp. 241–252.

[5] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. "Anomaly-based network intrusion detection: Techniques, systems and challenges". In: *Computers & Security* 28.1 (2009), pp. 18–28. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2008.08.003. URL: https://www.sciencedirect.com/science/article/pii/S0167404808000692.

[6] Jiong Zhang and Mohammad Zulkernine. "Anomaly Based Network Intrusion Detection with Unsupervised Outlier Detection". In: *2006 IEEE International Conference on Communications*. Vol. 5. 2006, pp. 2388–2393. DOI: 10.1109/ICC.2006.255127.

[7] Zhen Yang, Xiaodong Liu, Tong Li, Di Wu, Jinjiang Wang, Yunwei Zhao, and Han Han. "A systematic literature review of methods and datasets for anomaly-based network intrusion detection". In: *Computers & Security* 116 (2022), p. 102675. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2022.102675. URL: https://www.sciencedirect.com/science/article/pii/S0167404822000736.

[8] Robert M Lee. "2020 SANS Cyber Threat Intelligence (CTI) Survey". In: (2020).

[9] Haiguang Lai, Shengwen Cai, Hao Huang, Junyuan Xie, and Hui Li. "A Parallel Intrusion Detection System for High-Speed Networks". In: *Applied Cryptography and Network Security*. Ed. by Markus Jakobsson, Moti Yung, and Jianying Zhou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 439–451. ISBN: 978-3-540-24852-1.

[10] Ming Gao, Kenong Zhang, and Jiahua Lu. "Efficient packet matching for gigabit network intrusion detection using TCAMs". In: *20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*. Vol. 1. 2006, 6 pp.–254. DOI: 10.1109/AINA.2006.164.

[11] Muhammad Fahad Umer, Muhammad Sher, and Yaxin Bi. "Flow-based intrusion detection: Techniques and challenges". In: *Computers & Security* 70 (2017), pp. 238–254. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2017.05.009. URL: https://www.sciencedirect.com/science/article/pii/S0167404817301165.

[12] Zied Aouini and Adrian Pekar. "NFStream: A flexible network data analysis framework". In: *Computer Networks* 204 (2022), p. 108719. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.108719. URL: https://www.sciencedirect.com/science/article/pii/S1389128621005739.

[13] Anna Sperotto, Gregor Schaffrath, Ramin Sadre, Cristian Morariu, Aiko Pras, and Burkhard Stiller. "An Overview of IP Flow-Based Intrusion Detection". In: *IEEE Communications Surveys & Tutorials* 12.3 (2010), pp. 343–356. DOI: 10.1109/SURV.2010.032210.00054.

[14] Hongyu Liu and Bo Lang. "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey". In: *Applied Sciences* 9.20 (2019). ISSN: 2076-3417. DOI: 10.3390/app9204396. URL: https://www.mdpi.com/2076-3417/9/20/4396.

[15] Sultan Zavrak and Murat İskefiyeli. “Anomaly-Based Intrusion Detection From Network Flow Features Using Variational Autoencoder”. In: *IEEE Access* 8 (2020), pp. 108346–108358. DOI: `10.1109/ACCESS.2020.3001350`.

[16] Fahimeh Farahnakian and Jukka Heikkonen. “A deep auto-encoder based approach for intrusion detection system”. In: *2018 20th International Conference on Advanced Communication Technology (ICACT)*. 2018, pp. 178–183. DOI: `10.23919/ICACT.2018.8323688`.

[17] Lukas Budach, Moritz Feuerpfeil, Nina Ihde, Andrea Nathansen, Nele Noack, Hendrik Patzlaff, Felix Naumann, and Hazar Harmouch. *The Effects of Data Quality on Machine Learning Performance*. 2022. arXiv: `2207.14529 [cs.DB]`.

[18] Inês Martins, João S. Resende, Patrícia R. Sousa, Simão Silva, Luís Antunes, and João Gama. “Host-based IDS: A review and open issues of an anomaly detection system in IoT”. In: *Future Generation Computer Systems* 133 (2022), pp. 95–113. ISSN: 0167-739X. DOI: `https://doi.org/10.1016/j.future.2022.03.001`. URL: `https://www.sciencedirect.com/science/article/pii/S0167739X22000760`.

[19] Satish Kumar, Sunanda Gupta, and Sakshi Arora. “Research Trends in Network-Based Intrusion Detection Systems: A Review”. In: *IEEE Access* 9 (2021), pp. 157761–157779. DOI: `10.1109/ACCESS.2021.3129775`.

[20] D. Mangla and H. Gupta. “Application based intrusion detection system”. In: 9 (Jan. 2016), pp. 391–397.

[21] Tamer AbuHmed, Abedelaziz Mohaisen, and DaeHun Nyang. “A survey on deep packet inspection for intrusion detection systems”. In: *arXiv preprint arXiv:0803.0037* (2008).

[22] Holger Dreger, Anja Feldmann, Vern Paxson, and Robin Sommer. “Operational experiences with high-volume network intrusion detection”. In: *Proceedings of the 11th ACM conference on Computer and communications security*. 2004, pp. 2–11.

[23] Robert Koch. “Towards next-generation Intrusion Detection”. In: *2011 3rd International Conference on Cyber Conflict*. 2011, pp. 1–18.

[24] Benoit Claise, Brian Trammell, and Paul Aitken. *Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information*. Tech. rep. 2013.

[25] Benoit Claise. *Cisco systems netflow services export version 9*. Tech. rep. 2004.

[26] Simon Leinen. *Evaluation of candidate protocols for IP flow information export (IPFIX)*. Tech. rep. 2004.

[27] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. "A survey of network-based intrusion detection data sets". In: *Computers & Security* 86 (2019), pp. 147–167. ISSN: 0167-4048. DOI: `https://doi.org/10.1016/j.cose.2019.06.005`. URL: `https://www.sciencedirect.com/science/article/pii/S016740481930118X`.

[28] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Y. Sheng. "Malware traffic classification using convolutional neural network for representation learning". In: Jan. 2017, pp. 712–717. DOI: `10.1109/ICOIN.2017.7899588`.

[29] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: Jan. 2018, pp. 108–116. DOI: `10.5220/0006639801080116`.

[30] Nour Moustafa and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)*. 2015, pp. 1–6. DOI: `10.1109/MilCIS.2015.7348942`.

[31] Heather Lawrence, Uchenna Ezeobi, Orly Tauil, Jacob Nosal, Owen Redwood, Yanyan Zhuang, and Gedare Bloom. "CUPID: A labeled dataset with Pentesting for evaluation of network intrusion detection". In: *Journal of Systems Architecture* 129 (2022), p. 102621.

[32] Hanan Hindy, David Brosset, Ethan Bayne, Amar Kumar Seeam, Christos Tachtatzis, Robert Atkinson, and Xavier Bellekens. "A Taxonomy of Network Threats and the Effect of Current Datasets on Intrusion Detection Systems". In: *IEEE Access* 8 (2020), pp. 104650–104675. DOI: `10.1109/ACCESS.2020.3000179`.

[33] Mohanad Sarhan, Siamak Layeghy, Nour Moustafa, and Marius Portmann. "NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems". In: *Big Data Technologies and Applications*. Ed. by Zeng Deze, Huan Huang, Rui Hou, Seungmin Rho, and Naveen Chilamkurti. Cham: Springer International Publishing, 2021, pp. 117–135. ISBN: 978-3-030-72802-1.

[34] Gints Engelen, Vera Rimmer, and Wouter Joosen. “Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study”. In: *2021 IEEE Security and Privacy Workshops (SPW)*. 2021, pp. 7–12. DOI: 10.1109/SPW53761.2021.00009.

[35] Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, and Wouter Joosen. “Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018”. In: *2022 IEEE Conference on Communications and Network Security (CNS)*. 2022, pp. 254–262. DOI: 10.1109/CNS56114.2022.9947235.

[36] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[37] Mohamed Alloghani, Dhiya Al-Jumeily, Jamila Mustafina, Abir Hussain, and Ahmed J. Aljaaf. “A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science”. In: *Supervised and Unsupervised Learning for Data Science*. Ed. by Michael W. Berry, Azlinah Mohamed, and Bee Wah Yap. Cham: Springer International Publishing, 2020, pp. 3–21. ISBN: 978-3-030-22475-2. DOI: 10.1007/978-3-030-22475-2_1. URL: https://doi.org/10.1007/978-3-030-22475-2_1.

[38] Jesper E Van Engelen and Holger H Hoos. “A survey on semi-supervised learning”. In: *Machine learning* 109.2 (2020), pp. 373–440.

[39] Yanwen Chong, Yun Ding, Qing Yan, and Shaoming Pan. “Graph-based semi-supervised learning: A review”. In: *Neurocomputing* 408 (2020), pp. 216–230. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.12.130. URL: https://www.sciencedirect.com/science/article/pii/S0925231220304938.

[40] Ningxin Shi, Xiaohong Yuan, Joaquin Hernandez, Kaushik Roy, and Albert Esterline. “Self-Learning Semi-Supervised Machine Learning for Network Intrusion Detection”. In: *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2018, pp. 59–64. DOI: 10.1109/CSCI46756.2018.00019.

[41] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. “Semi-supervised Learning with Deep Generative Models”. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. Vol. 27. Curran As-

sociates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/d523773c6b194f37b938d340d5d02232-Paper.pdf.

[42] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8 (1992), pp. 279–292.

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[44] Peter J. Rousseeuw. “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53–65. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. URL: https://www.sciencedirect.com/science/article/pii/0377042787901257.

[45] David L. Davies and Donald W. Bouldin. “A Cluster Separation Measure”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1.2 (1979), pp. 224–227. DOI: 10.1109/TPAMI.1979.4766909.

[46] Leo Breiman. “Random forests”. In: *Machine learning* 45 (2001), pp. 5–32.

[47] Pierre Geurts, Damien Ernst, and Louis Wehenkel. “Extremely randomized trees”. In: *Machine learning* 63 (2006), pp. 3–42.

[48] Junyi Chai, Hao Zeng, Anming Li, and Eric W.T. Ngai. “Deep learning in computer vision: A critical review of emerging techniques and application scenarios”. In: *Machine Learning with Applications* 6 (2021), p. 100134. ISSN: 2666-8270. DOI: https://doi.org/10.1016/j.mlwa.2021.100134. URL: https://www.sciencedirect.com/science/article/pii/S2666827021000670.

[49] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. “A Survey of the Usages of Deep Learning for Natural Language Processing”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.2 (2021), pp. 604–624. DOI: 10.1109/TNNLS.2020.2979670.

[50] Fionn Murtagh. “Multilayer perceptrons for classification and regression”. In: *Neurocomputing* 2.5 (1991), pp. 183–197. ISSN: 0925-2312. DOI: https://doi.org/10.1016/0925-2312(91)90023-5. URL: https://www.sciencedirect.com/science/article/pii/0925231291900235.

[51] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML’10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077.

[52] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: `https://proceedings.mlr.press/v15/glorot11a.html`.

[53] Pierre Baldi. “Autoencoders, Unsupervised Learning, and Deep Architectures”. In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. Ed. by Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver. Vol. 27. Proceedings of Machine Learning Research. Bellevue, Washington, USA: PMLR, Feb. 2012, pp. 37–49. URL: `https://proceedings.mlr.press/v27/baldi12a.html`.

[54] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*. 2020. DOI: `10.48550/ARXIV.2003.05991`. URL: `https://arxiv.org/abs/2003.05991`.

[55] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014. URL: `http://arxiv.org/abs/1312.6114`.

[56] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. “Adversarial Autoencoders”. In: *CoRR* abs/1511.05644 (2015). arXiv: `1511.05644`. URL: `http://arxiv.org/abs/1511.05644`.

[57] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger. 2014, pp. 2672–2680. URL: `https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html`.

[58] 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, 20–22 Apr 2017, pp. 1273–1282. URL: `https://proceedings.mlr.press/v54/mcmahan17a.html`.

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

[60] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. *On the Convergence of FedAvg on Non-IID Data*. 2019. arXiv: `1907.02189 [stat.ML]`.

[61] 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*. Ed. by I. Dhillon, D. Papailiopoulos, and V. Sze. Vol. 2. 2020, pp. 429–450. URL: `https://proceedings.mlsys.org/paper/2020/file/38af86134b65d0f10fe33d30dd76442e-Paper.pdf`.

[62] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris S. Papailiopoulos, and Yasaman Khazaeni. "Federated Learning with Matched Averaging". In: *CoRR* abs/2002.06440 (2020). arXiv: `2002.06440`. URL: `https://arxiv.org/abs/2002.06440`.

[63] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. "Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization". In: *CoRR* abs/2007.07481 (2020). arXiv: `2007.07481`. URL: `https://arxiv.org/abs/2007.07481`.

[64] Xiaoxiao Li, Meirui JIANG, Xiaofei Zhang, Michael Kamp, and Qi Dou. "Fed{BN}: Federated Learning on Non-{IID} Features via Local Batch Normalization". In: *International Conference on Learning Representations*. 2021. URL: `https://openreview.net/forum?id=6YEQUn0QICG`.

[65] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. “Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training”. In: *CoRR* abs/1712.01887 (2017). arXiv: `1712.01887`. URL: `http://arxiv.org/abs/1712.01887`.

[66] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. “Federated Learning: Strategies for Improving Communication Efficiency”. In: *CoRR* abs/1610.05492 (2016). arXiv: `1610.05492`. URL: `http://arxiv.org/abs/1610.05492`.

[67] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. “Practical Secure Aggregation for Privacy-Preserving Machine Learning”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1175–1191. ISBN: 9781450349468. DOI: `10.1145/3133956.3133982`. URL: `https://doi.org/10.1145/3133956.3133982`.

[68] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. “Towards Federated Learning at Scale: System Design”. In: *CoRR* abs/1902.01046 (2019). arXiv: `1902.01046`. URL: `http://arxiv.org/abs/1902.01046`.

[69] Jeffrey Li, Mikhail Khodak, Sebastian Caldas, and Ameet Talwalkar. “Differentially Private Meta-Learning”. In: *CoRR* abs/1909.05830 (2019). arXiv: `1909.05830`. URL: `http://arxiv.org/abs/1909.05830`.

[70] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. “Federated Multi-Task Learning”. In: *CoRR* abs/1705.10467 (2017). arXiv: `1705.10467`. URL: `http://arxiv.org/abs/1705.10467`.

[71] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D’Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Fari-

naz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. "Advances and Open Problems in Federated Learning". In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210. ISSN: 1935-8237. DOI: 10.1561/2200000083. URL: http://dx.doi.org/10.1561/2200000083.

[72] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. "Federated Learning of Deep Networks using Model Averaging". In: *CoRR* abs/1602.05629 (2016). arXiv: 1602.05629. URL: http://arxiv.org/abs/1602.05629.

[73] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. "Federated Learning with Non-IID Data". In: *CoRR* abs/1806.00582 (2018). arXiv: 1806.00582. URL: http://arxiv.org/abs/1806.00582.

[74] Eunjeong Jeong, Seungeun Oh, Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. "Communication-Efficient On-Device Machine Learning: Federated Distillation and Augmentation under Non-IID Private Data". In: *CoRR* abs/1811.11479 (2018). arXiv: 1811.11479. URL: http://arxiv.org/abs/1811.11479.

[75] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. "Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies". In: *CoRR* abs/2010.01243 (2020). arXiv: 2010.01243. URL: https://arxiv.org/abs/2010.01243.

[76] Yae Jee Cho, Samarth Gupta, Gauri Joshi, and Osman Yagan. "Bandit-based Communication-Efficient Client Selection Strategies for Federated Learning". In: *CoRR* abs/2012.08009 (2020). arXiv: 2012.08009. URL: https://arxiv.org/abs/2012.08009.

[77] Wenlin Chen, Samuel Horvath, and Peter Richtárik. "Optimal Client Sampling for Federated Learning". In: *CoRR* abs/2010.13723 (2020). arXiv: 2010.13723. URL: https://arxiv.org/abs/2010.13723.

[78] Mónica Ribero and Haris Vikalo. “Communication-Efficient Federated Learning via Optimal Client Sampling”. In: *CoRR* abs/2007.15197 (2020). arXiv: 2007.15197. URL: https://arxiv.org/abs/2007.15197.

[79] Arwa Aldweesh, Abdelouahid Derhab, and Ahmed Z. Emam. “Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues”. In: *Knowledge-Based Systems* 189 (2020), p. 105124. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2019.105124. URL: https://www.sciencedirect.com/science/article/pii/S0950705119304897.

[80] Shuhana Azmin and A. B. M Alim Al Islam. “Network Intrusion Detection System Based on Conditional Variational Laplace AutoEncoder”. In: *Proceedings of the 7th International Conference on Networking, Systems and Security*. NSysS ’20. Dhaka, Bangladesh: Association for Computing Machinery, 2020, pp. 82–88. ISBN: 9781450389051. DOI: 10.1145/3428363.3428371. URL: https://doi.org/10.1145/3428363.3428371.

[81] Dongyang Li, Daisuke Kotani, and Yasuo Okabe. “Improving Attack Detection Performance in NIDS Using GAN”. In: *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2020, pp. 817–825. DOI: 10.1109/COMPSAC48688.2020.0-162.

[82] Dashun Liao, Sunpei Huang, Yuyu Tan, and Guoqing Bai. “Network Intrusion Detection Method Based on GAN Model”. In: *2020 International Conference on Computer Communication and Network Security (CCNS)*. 2020, pp. 153–156. DOI: 10.1109/CCNS50731.2020.00041.

[83] Zilong Lin, Yong Shi, and Zhi Xue. “Idsgan: Generative adversarial networks for attack generation against intrusion detection”. In: *Pacific-asia conference on knowledge discovery and data mining*. Springer. 2022, pp. 79–91.

[84] Giuseppina Andresini, Annalisa Appice, Luca De Rose, and Donato Malerba. “GAN augmentation to deal with imbalance in imaging-based intrusion detection”. In: *Future Generation Computer Systems* 123 (2021), pp. 108–127. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2021.04.017. URL: https://www.sciencedirect.com/science/article/pii/S0167739X21001382.

[85] Meryem Janati Idrissi, Hamza Alami, Abdelkader El Mahdaouy, Abdellah El Mekki, Soufiane Oualil, Zakaria Yartaoui, and Ismail Berrada. “Fed-ANIDS: Federated learning for anomaly-based network intrusion detection systems”. In: *Expert Systems with Applications* 234 (2023), p. 121000. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.121000. URL: https://www.sciencedirect.com/science/article/pii/S0957417423015026.

[86] Aliya Tabassum, Aiman Erbad, Wadha Lebda, Amr Mohamed, and Mohsen Guizani. “FEDGAN-IDS: Privacy-preserving IDS using GAN and Federated Learning”. In: *Computer Communications* 192 (2022), pp. 299–310. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2022.06.015. URL: https://www.sciencedirect.com/science/article/pii/S0140366422002171.

[87] Laurent Dinh, David Krueger, and Yoshua Bengio. “Nice: Non-linear independent components estimation”. In: *arXiv preprint arXiv:1410.8516* (2014).

[88] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using Real NVP”. In: *CoRR* abs/1605.08803 (2016). arXiv: 1605.08803. URL: http://arxiv.org/abs/1605.08803.

[89] Ev Zisselman and Aviv Tamar. “Deep Residual Flow for Out of Distribution Detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[90] Meryem Janati Idrissi, Hamza Alami, Abdelhak Bouayad, and Ismail Berrada. “NF-NIDS: Normalizing Flows for Network Intrusion Detection Systems”. In: *2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM)*. 2023, pp. 1–7. DOI: 10.1109/WINCOM59760.2023.10322987.

[91] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin. “Intrusion detection by machine learning: A review”. In: *Expert Systems with Applications* 36.10 (2009), pp. 11994–12000. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2009.05.029. URL: https://www.sciencedirect.com/science/article/pii/S0957417409004801.

[92] Nazli Tekin, Abbas Acar, Ahmet Aris, A. Selcuk Uluagac, and Vehbi Cagri Gungor. “Energy consumption of on-device machine learning models for IoT intrusion detection”. In: *Internet of Things* 21 (2023), p. 100670. ISSN: 2542-

6605. DOI: https://doi.org/10.1016/j.iot.2022.100670. URL: https://www.sciencedirect.com/science/article/pii/S2542660522001512.

[93] Sawsan Abdul Rahman, Hanine Tout, Chamseddine Talhi, and Azzam Mourad. "Internet of Things Intrusion Detection: Centralized, On-Device, or Federated Learning?" In: *IEEE Network* 34.6 (2020), pp. 310–317. DOI: 10.1109/MNET.011.2000286.

[94] Thi-Nga Dao and HyungJune Lee. "JointNIDS: Efficient Joint Traffic Management for On-Device Network Intrusion Detection". In: *IEEE Transactions on Vehicular Technology* 71.12 (2022), pp. 13254–13265. DOI: 10.1109/TVT.2022.3198266.

[95] Shaashwat Agrawal, Sagnik Sarkar, Ons Aouedi, Gokul Yenduri, Kandaraj Piamrat, Mamoun Alazab, Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, and Thippa Reddy Gadekallu. "Federated Learning for intrusion detection system: Concepts, challenges and future directions". In: *Computer Communications* 195 (2022), pp. 346–361. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2022.09.012. URL: https://www.sciencedirect.com/science/article/pii/S0140366422003516.

[96] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N. Asokan, and Ahmad-Reza Sadeghi. "DIoT: A Federated Self-learning Anomaly Detection System for IoT". In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2019, pp. 756–767. DOI: 10.1109/ICDCS.2019.00080.

[97] 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.

[98] Pu Tian, Zheyi Chen, Wei Yu, and Weixian Liao. "Towards asynchronous federated learning based threat detection: A DC-Adam approach". In: *Computers & Security* 108 (2021), p. 102344. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2021.102344. URL: https://www.sciencedirect.com/science/article/pii/S0167404821001681.

[99] Yann LeCun, Corinna Cortes, and Chris Burges. *MNIST handwritten digit database. 2010.* http://yann.lecun.com/exdb/mnist.

[100] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. "Toward generating a new intrusion detection dataset and intrusion traffic characterization." In: *ICISSp* 1 (2018), pp. 108–116.

[101] S Garcia, A Parmisano, and MJ Erquiaga. *IoT-23: A Labeled Dataset with Malicious and Benign IoT Network Traffic. 2020.* `http://doi.org/10.5281/zenodo.4743746`.

[102] 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), p. 108693. ISSN: 1389-1286. DOI: `https://doi.org/10.1016/j.comnet.2021.108693`. URL: `https://www.sciencedirect.com/science/article/pii/S1389128621005582`.

[103] 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`.

[104] 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, pp. 273–279. ISBN: 9781450372459. DOI: `10.1145/3368926.3369705`. URL: `https://doi.org/10.1145/3368926.3369705`.

[105] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. "Characterization of Encrypted and VPN Traffic Using Time-Related Features". In: Feb. 2016. DOI: `10.5220/0005740704070414`.

[106] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. "Characterization of Tor Traffic using Time based Features". In: Jan. 2017, pp. 253–262. DOI: `10.5220/0006105602530262`.

[107] Mohamed Ali Ayed and Chamseddine Talhi. "Federated Learning for Anomaly-Based Intrusion Detection". In: *2021 International Symposium on Networks, Computers and Communications (ISNCC)*. 2021, pp. 1–8. DOI: `10.1109/ISNCC52172.2021.9615816`.

[108] Yang Qin and Masaaki Kondo. "Federated Learning-Based Network Intrusion Detection with a Feature Selection Approach". In: *2021 International Conference*

on Electrical, Communication, and Computer Engineering (ICECCE). 2021, pp. 1–6. DOI: 10.1109/ICECCE52056.2021.9514222.

[109] Mineto Tsukada, Masaaki Kondo, and Hiroki Matsutani. "A Neural Network-Based On-Device Learning Anomaly Detector for Edge Devices". In: *IEEE Transactions on Computers* 69.7 (2020), pp. 1027–1044. DOI: 10.1109/TC.2020.2973631.

[110] *KDD Cup 1999 Data.* Available online https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed: 2024-04-24.

[111] Nour Moustafa and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)*. 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

[112] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection". In: *CoRR* abs/1802.09089 (2018). arXiv: 1802.09089. URL: http://arxiv.org/abs/1802.09089.

[113] Majjed Al-Qatf, Yu Lasheng, Mohammed Al-Habib, and Kamal Al-Sabahi. "Deep Learning Approach Combining Sparse Autoencoder With SVM for Network Intrusion Detection". In: *IEEE Access* 6 (2018), pp. 52843–52856. DOI: 10.1109/ACCESS.2018.2869577.

[114] Gints Engelen, Vera Rimmer, and Wouter Joosen. "Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study". In: *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2021, pp. 7–12.

[115] Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, and Wouter Joosen. "Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018". In: *2022 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2022, pp. 254–262.

[116] 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.

[117] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. "Adversarial Feature Learning". In: *CoRR* abs/1605.09782 (2016). arXiv: 1605.09782. URL: http://arxiv.org/abs/1605.09782.

[118] Federico Di Mattia, Paolo Galeone, Michele De Simoni, and Emanuele Ghelfi. "A Survey on GANs for Anomaly Detection". In: *CoRR* abs/1906.11632 (2019). arXiv: `1906.11632`. URL: `http://arxiv.org/abs/1906.11632`.

[119] Camila F. T. Pontes, Manuela M. C. de Souza, João J. C. Gondim, Matt Bishop, and Marcelo Antonio Marotta. "A New Method for Flow-Based Network Intrusion Detection Using the Inverse Potts Model". In: *IEEE Transactions on Network and Service Management* 18.2 (2021), pp. 1125–1136. DOI: `10.1109/TNSM.2021.3075503`.

[120] Loc Gia Nguyen and Kohei Watabe. "Flow-Based Network Intrusion Detection Based on BERT Masked Language Model". In: *Proceedings of the 3rd International CoNEXT Student Workshop*. CoNEXT-SW '22. Rome, Italy: Association for Computing Machinery, 2022, pp. 7–8. ISBN: 9781450399371. DOI: `10.1145/3565477.3569152`. URL: `https://doi.org/10.1145/3565477.3569152`.

[121] V. Jyothsna, D. Mukesh, and A. N. Sreedhar. "A Flow-Based Network Intrusion Detection System for High-Speed Networks Using Meta-heuristic Scale". In: *Computing and Network Sustainability*. Ed. by Sheng-Lung Peng, Nilanjan Dey, and Mahesh Bundele. Singapore: Springer Singapore, 2019, pp. 337–347. ISBN: 978-981-13-7150-9.

[122] Ganesh Sadasivan, Nevil Brownlee, Benoit Claise, and Juergen Quittek. *Architecture for IP flow information export*. Tech. rep. 2009.

[123] Palo Alto Networks. Accessed: 2023-11-17. 2005.

[124] Fortinet. Accessed: 2023-11-17. 2000.

[125] Adrian Pekar, Alejandra Duque-Torres, Winston Seah, and Mauricio Caicedo. "Knowledge Discovery: Can It Shed New Light on Threshold Definition for Heavy-Hitter Detection?" In: *Journal of Network and Systems Management* 29 (July 2021), p. 30. DOI: `10.1007/s10922-021-09593-w`.

[126] Lan Liu, Pengcheng Wang, Jianliang Ruan, and Jun Lin. *ConFlow: Contrast Network Flow Improving Class-Imbalanced Learning in Network Intrusion Detection*. Apr. 2022. DOI: `10.21203/rs.3.rs-1572776/v1`.

[127] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek

Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. "TensorFlow: A System for Large-Scale Machine Learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*. Ed. by Kimberly Keeton and Timothy Roscoe. USENIX Association, 2016, pp. 265–283. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[128] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett. 2019, pp. 8024–8035. URL: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[129] Mohanad Sarhan, Siamak Layeghy, Nour Moustafa, and Marius Portmann. "Towards a Standard Feature Set of NIDS Datasets". In: *CoRR* abs/2101.11315 (2021). arXiv: 2101.11315. URL: https://arxiv.org/abs/2101.11315.

[130] Ofek Bader, Adi Lichy, Amit Dvir, Ran Dubin, and Chen Hajaj. "Open-Source Framework for Encrypted Internet and Malicious Traffic Classification". In: *arXiv preprint arXiv:2206.10144* (2022).

[131] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. "Scikit-learn: Machine Learning in Python". In: *J. Mach. Learn. Res.* 12 (2011), pp. 2825–2830. DOI: 10.5555/1953048.2078195. URL: https://dl.acm.org/doi/10.5555/1953048.2078195.

[132] Muhammad Asad, Ahmed Moustafa, Takayuki Ito, and Muhammad Aslam. "Evaluating the Communication Efficiency in Federated Learning Algorithms". In: *2021 IEEE 24th International Conference on Computer Supported*

Cooperative Work in Design (CSCWD). 2021, pp. 552–557. DOI: 10.1109/CSCWD49262.2021.9437738.

[133] Thien Duc Nguyen, Phillip Rieger, Markus Miettinen, and Ahmad-Reza Sadeghi. "Poisoning Attacks on Federated Learning-based IoT Intrusion Detection System". In: 2020. URL: https://api.semanticscholar.org/CorpusID:216086694.

[134] Angelo Feraudo, Poonam Yadav, Vadim Safronov, Diana Andreea Popescu, Richard Mortier, Shiqiang Wang, Paolo Bellavista, and Jon Crowcroft. "CoLearn: Enabling Federated Learning in MUD-Compliant IoT Edge Networks". In: *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*. EdgeSys '20. Heraklion, Greece: Association for Computing Machinery, 2020, pp. 25–30. ISBN: 9781450371322. DOI: 10.1145/3378679.3394528. URL: https://doi.org/10.1145/3378679.3394528.

[135] Sawsan Abdulrahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. "A Survey on Federated Learning: The Journey From Centralized to Distributed On-Site Learning and Beyond". In: *IEEE Internet of Things Journal* 8.7 (2021), pp. 5476–5497. DOI: 10.1109/JIOT.2020.3030072.

[136] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. "Federated Learning with Differential Privacy: Algorithms and Performance Analysis". In: *IEEE Transactions on Information Forensics and Security* 15 (2020). Cited by: 707; All Open Access, Bronze Open Access, Green Open Access, pp. 3454–3469. DOI: 10.1109/TIFS.2020.2988575. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083775015&doi=10.1109%2fTIFS.2020.2988575&partnerID=40&md5=70e863d48fd39efdbd36959265772c95.

[137] Chuan Zhao, Shengnan Zhao, Minghao Zhao, Zhenxiang Chen, Chong-Zhi Gao, Hongwei Li, and Yu-an Tan. "Secure Multi-Party Computation: Theory, practice and applications". In: *Information Sciences* 476 (2019), pp. 357–372. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2018.10.024. URL: https://www.sciencedirect.com/science/article/pii/S0020025518308338.

[138] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. "BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning". In: *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, July 2020, pp. 493–506. ISBN: 978-1-939133-14-4. URL: `https://www.usenix.org/conference/atc20/presentation/zhang-chengliang`.

[139] Beibei Li, Yuhao Wu, Jiarui Song, Rongxing Lu, Tao Li, and Liang Zhao. "DeepFed: Federated Deep Learning for Intrusion Detection in Industrial Cyber–Physical Systems". In: *IEEE Transactions on Industrial Informatics* 17.8 (2021), pp. 5615–5624. DOI: `10.1109/TII.2020.3023430`.

[140] Ajesh Koyatan Chathoth, Abhyuday Jagannatha, and Stephen Lee. "Federated Intrusion Detection for IoT with Heterogeneous Cohort Privacy". In: *CoRR* abs/2101.09878 (2021). arXiv: `2101.09878`. URL: `https://arxiv.org/abs/2101.09878`.

[141] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. "Federated learning on non-IID data: A survey". In: *Neurocomputing* 465 (2021), pp. 371–390. ISSN: 0925-2312. DOI: `https://doi.org/10.1016/j.neucom.2021.07.098`. URL: `https://www.sciencedirect.com/science/article/pii/S0925231221013254`.

[142] Xiaodong Ma, Jia Zhu, Zhihao Lin, Shanxuan Chen, and Yangjie Qin. "A state-of-the-art survey on solving non-IID data in Federated Learning". In: *Future Generation Computer Systems* 135 (2022), pp. 244–258. ISSN: 0167-739X. DOI: `https://doi.org/10.1016/j.future.2022.05.003`. URL: `https://www.sciencedirect.com/science/article/pii/S0167739X22001686`.

[143] Sergey Ioffe. "Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models". In: *CoRR* abs/1702.03275 (2017). arXiv: `1702.03275`. URL: `http://arxiv.org/abs/1702.03275`.

[144] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B. Gibbons. *The Non-IID Data Quagmire of Decentralized Machine Learning*. 2019. arXiv: `1910.00189 [cs.LG]`.

[145] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto* (May 2012).

[146] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[147] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *CoRR* abs/1708.07747 (2017). arXiv: 1708.07747. URL: http://arxiv.org/abs/1708.07747.

[148] Gozde Karatas, Onder Demir, and Ozgur Koray Sahingoz. “Increasing the Performance of Machine Learning-Based IDSs on an Imbalanced and Up-to-Date Dataset”. In: *IEEE Access* 8 (2020), pp. 32150–32162. DOI: 10.1109/ACCESS.2020.2973219.

[149] Yanpei Hua. “An Efficient Traffic Classification Scheme Using Embedded Feature Selection and LightGBM”. In: *2020 Information Communication Technologies Conference (ICTC)*. 2020, pp. 125–130. DOI: 10.1109/ICTC49638.2020.9123302.

Appendix

8.3 Features

We provide the set of features presented in both works [129] and [130]. Table 8.1 presents the features selected in the paper [129]. Table 8.2 describes the flow features we used and presented in [130].

Table 8.1: Set of features proposed in [129].

Feature	**Description**
tcp_flags	Cumulative of all flow TCP flags
src2dst_flags	Cumulative of all client TCP flags
dst2src_flags	Cumulative of all server TCP flags
min_ttl	Min flow TTL
max_ttl	Max flow TTL
min_ip_pkt_len	Len of the smallest flow IP packet observed
max_ip_pkt_len	Len of the largest flow IP packet observed
src_to_dst_second_bytes	Bytes/sec (src->dst)
dst_to_src_second_bytes	Bytes/sec (dst->src)
retransmitted_in_bytes	Number of retransmitted TCP flow bytes (src->dst)
retransmitted_in_packets	Number of retransmitted TCP flow packets (src->dst)
retransmitted_out_bytes	Number of retransmitted TCP flow bytes (dst->src)
retransmitted_out_packets	Number of retransmitted TCP flow packets (dst->src)
src_to_dst_avg_throughput	Src to dst average that (bps)
dst_to_src_avg_throughput	Dst to src average that (bps)
num_pkts_up_to_128_bytes	packets whose IP size <= 128
num_pkts_128_to_256_bytes	packets whose IP size > 128 and <= 256
num_pkts_256_to_512_bytes	packets whose IP size > 256 and < 512
num_pkts_512_to_1024_bytes	packets whose IP size > 512 and < 1024
num_pkts_1024_to_1514_bytes	packets whose IP size > 1024 and <= 1514
tcp_win_max_in	Max TCP Window (src->dst)
tcp_win_max_out	Max TCP Window (dst->src)
icmp_type	ICMP Type * 256 + ICMP code
icmp_v4_type	ICMP Type
dns_query_id	DNS query transaction Id
dns_query_type	"DNS query type (e.g., 1=A, 2=NS..)"
dns_ttl_answer	TTL of the first A record (if any)
ftp_command_ret_code	FTP client command return code

Table 8.2: Set of features proposed in [130].

Features	Description
src2dst_first_packet_payload_len	First packet's payload length src2dst direction
dst2src_first_packet_payload_len	First packet's payload length dst2src direction
src2dst_most_freq_payload_ratio	The ratio of number of packets with most freq payload len for direction src2dst to the the total number of packets in direction src2dst
src2dst_most_freq_payload_len	The number of packets with most freq payload len for direction src2dst
dst2src_most_freq_payload_ratio	The ratio of number of packets with most freq payload len for direction dst2src to the the total number of packets in direction src2dst
dst2src_most_freq_payload_len	The number of packets with most freq payload len for direction dst2src
bidirectional_mean_packet_relative_times	The average timestamp in milliseconds between first flow bidirectional packet and all other flow bidirectional packets.
bidirectional_stddev_packet_relative_times	The standard deviation timestamp in milliseconds between first flow bidirectional packet and all other flow bidirectional packets.

Continued on next page

Table 8.2 – continued from previous page

Features	Description
bidirectional_variance_packet_relative_times	The variance timestamp in milliseconds between first flow bidirectional packet and all other flow bidirectional packets.
bidirectional_coeff_of_var_packet_relative_times	The coefficient of variance (std/avg) of the difference between first flow bidirectional packet and all other flow bidirectional packets.
bidirectional_skew_from_median_packet_relative_times	The skewness from median (3$\times$(avg-median) / std) of the difference between first flow bidirectional packet and all other flow bidirectional packets.
src2dst_mean_packet_relative_times	The average timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction src->dst.
src2dst_stddev_packet_relative_times	The standard deviation timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction src->dst.
src2dst_variance_packet_relative_times	The variance timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction src->dst.
src2dst_coeff_of_var_packet_relative_times	The coefficient of variance (std/avg) of the difference between first flow bidirectional packet and all other flow packets with direction src->dst.

Continued on next page

Table 8.2 – continued from previous page

Features	Description
src2dst_skew_from_median_packet_relative_times	The skewness from median (3×(avg-median) / std) of the difference between first flow bidirectional packet and all other flow packets with direction src->dst.
dst2src_mean_packet_relative_times	The average timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction dst->src.
dst2src_stddev_packet_relative_times	The standard deviation timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction dst->src.
dst2src_variance_packet_relative_times	The variance timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction dst->src.
dst2src_coeff_of_var_packet_relative_times	The coefficient of variance (std/avg) of the difference between first flow bidirectional packet and all other flow packets with direction dst->src.
dst2src_skew_from_median_packet_relative_times	The skewness from median (3×(avg-median) / std) of the difference between first flow bidirectional packet and all other flow packets with direction dst->src.

Continued on next page

Table 8.2 – continued from previous page

Features	Description
min_req_res_time_diff	The minimum timestamp difference between the request and its response (delta time between packet from source and the next packet from dst or vice versa).
max_req_res_time_diff	The maximum timestamp difference between the request and its response (delta time between packet from source and the next packet from dst or vice versa)
mean_req_res_time_diff	The mean of all timestamp difference between the request and its response (delta time between packet from source and the next packet from dst or vice versa)
median_req_res_time_diff	The median of all timestamp difference between the request and its response (delta time between packet from source and the next packet from dst or vice versa)
stddev_req_res_time_diff	The standard deviation of all timestamp difference between the request and its response (timestamp between packet from source and the next packet from dst or vice versa)
variance_req_res_time_diff	The variance of all timestamp difference between the request and its response (timestamp between packet from source and the next packet from dst or vice versa)

Continued on next page

Table 8.2 – continued from previous page

Features	Description
coeff_of_var_req_res_time_diff	The coefficient of variance (std/avg) of all timestamp difference between the request and its response (timestamp between packet from source and the next packet from dst or vice versa).
skew_from_median_req_res_time_diff	The skewness from median (3×(avg-median) / std) of all timestamp difference between the request and its response (timestamp between packet from source and the next packet from dst or vice versa).
src2dst_small_packet_payload_packets	The number of src2dst packets with a small size (< 32 bytes)
src2dst_small_packet_payload_ratio	The ratio of small src2dst packets (<32 bytes) to the number of all src2dst packets.
dst2src_small_packet_payload_packets	The number of dst2src packets with a small size (< 32 bytes)
dst2src_small_packet_payload_ratio	The ratio of small dst2src packets (<32 bytes) to the number of all src2dst packets.
sent_recv_packet_ratio	The ratio of the number of received packets to the number of sent packets = recv/sent.
bidirectional_ps_first_quartile	The first quartile of flow packets size (link layer packet size).
bidirectional_ps_second_quartile	The second quartile of flow packets size (link layer packet size).
bidirectional_ps_third_quartile	The third quartile of flow packets size (link layer packet size).

Continued on next page

,

Table 8.2 – continued from previous page

Features	Description
bidirectional_ps_median_absoulte_deviation	The median of the absolute difference between packets size and the median (median × | packet_size - median |)
bidirectional_ps_skewness	The skewness of flow packets size (link layer packet size).
bidirectional_ps_kurtosis	The kurosis of flow packets size (link layer packet size).
bidirectional_piat_first_quartile	The first quartile of flow packets delta time (Delta time in milliseconds with previous flow packet).
bidirectional_piat_second_quartile	The second quartile of flow packets delta time (Delta time in milliseconds with previous flow packet).
bidirectional_piat_third_quartile	The third quartile of flow packets delta time (Delta time in milliseconds with previous flow packet).
bidirectional_piat_median_absoulte_deviation	The median of the absolute difference between flow packets delta time and the median (median × | packet_size - median |)
bidirectional_piat_skewness	The skewness of flow packets delta time.
bidirectional_piat_kurtosis	The kurtosis of flow packets delta time.