UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

Robust Context-Aware Intrusion Detection Systems through Personalised Federated Learning

Diogo Alexandre Nascimento Fernandes

Mestrado em Engenharia Informática

Dissertação orientada por:
Prof. Doutor Vinicius Vielmo Cogo
Prof. Doutor Nuno Fuentecilla Maia Ferreira Neves

2025

To my family, my girlfriend, and my friends

Acknowledgments

Firstly, I would like to express my deepest gratitude to my advisors, Prof. Dr. Vinicius Vielmo Cogo and Prof. Dr. Nuno Fuentecilla Maia Ferreira Neves, for their knowledge, guidance, and constant support throughout this research. Their insights and directions were invaluable in helping me develop a meaningful dissertation.

I also wish to dedicate this work to three very important people in my life: my mother, for her unwavering support, education, and motivation; my grandmother, who has been like a second mother and one of the most supportive figures in my life; and my girlfriend, whose encouragement and understanding sustained me through the most difficult moments of this journey.

Finally, I extend my sincere appreciation to all my friends and family, who were always by my side regardless of the circumstances. To each of them, without exception, I owe my deepest gratitude.

Resumo

A proliferação de dispositivos conectados em rede tem originado um aumento substancial da superfície de ataque e elevado a complexidade dos mecanismos de defesa contra ameaças cibernéticas. Os Sistemas de Deteção de Intrusões (IDS) assumem um papel central neste processo, uma vez que permitem monitorizar o tráfego e identificar atividades maliciosas. Contudo, as abordagens tradicionais, sobretudo as baseadas em coleções de assinaturas, apresentam limitações estruturais significativas: embora eficazes na deteção de ataques já catalogados, demonstram fraca capacidade de generalização perante ataques inéditos e dependem de atualizações frequentes de bases de dados. A introdução de técnicas de aprendizagem de máquina (ML) trouxe avanços consideráveis, mas estas soluções dependem frequentemente em conjunto de dados centralizados que levantam preocupações sérias relativamente à privacidade, ao consumo de recursos e à suscetibilidade a manipulações adversariais.

A Aprendizagem Federada (FL) surgiu como alternativa viável, permitindo que múltiplos participantes treinem modelos de forma colaborativa sem necessidade de centralizar os dados. Este paradigma contribui para a preservação da privacidade e distribui a carga computacional, mas a sua aplicação em IDS levanta desafios complexos: a heterogeneidade dos dados entre participantes, que compromete a convergência e a precisão dos modelos; a ausência de mecanismos de personalização, o que conduz a desempenhos díspares quando os dados locais apresentam elevada heterogeneidade entre os participantes; a vulnerabilidade a ataques de envenenamento (poisoning attacks), em que participantes maliciosos enviam atualizações manipuladas; e a problemas de escalabilidade em cenários com um número elevado de nós.

A presente dissertação propõe uma arquitetura híbrida de Aprendizagem Federada Personalizada para IDS em redes peer-to-peer (P2P), concebida para superar estas limitações. A solução articula três componentes fundamentais: uma fase inicial totalmente descentralizada, uma fase estruturada organizada num conjunto de topologias em árvore e um protocolo de reputação para gerir a confiança entre participantes. Na fase descentralizada, todos os participantes interagem de forma não estruturada, beneficiando da flexibilidade, redundância e tolerância a falhas. Posteriormente, a rede evolui para uma topologia organizada, em que os superpeers assumem a coordenação e agregação de modelos. A seleção destes superpeers é baseada em reputação, calculada continuamente a partir de métricas como a consistência entre rondas e contributo efetivo para o modelo global. Participantes maliciosos ou inconsistentes são penalizados, podendo ser excluídos do processo de treino, o que assegura resiliência e estabilidade do sistema.

O protocolo de reputação foi concebido de forma distribuída, atribuindo a cada participante a responsabilidade de avaliar os contributos dos seus vizinhos. Estas avaliações são propagadas pela rede através de um protocolo flooding, que assegura uma disseminação eficiente, tolerante a falhas, reduzindo a sobrecarga de comunicação e evitando pontos únicos de falha. A agregação destas avaliações permite identificar os participantes mais fiáveis, promovendo-os a superpeers e atribuindo-lhes papéis centrais na estrutura hierárquica, enquanto os menos confiáveis podem ser despromovidos. Esta dinâmica confere à rede elevada adaptabilidade, permitindo reconfigurações automáticas face à entrada ou saída de participantes, mantendo a robustez e o equilíbrio do sistema mesmo em ambientes distribuídos e heterogéneos.

Cada participante desempenha simultaneamente funções de cliente e de servidor: como cliente, treina localmente o modelo com os seus dados, aplicando personalização para alinhar o modelo global com a sua distribuição local; como servidor, gere conexões, participa em agregações e dissemina atualizações. Para assegurar estas operações e o correto funcionamento do sistema, foram desenvolvidos componentes específicos para a inicialização da rede, preparação e distribuição de dados, coordenação de rondas, eleição de superpeers, execução do protocolo de reputação e avaliação dos modelos.

A preparação experimental baseou-se no Kitsune Network Attack Dataset, que contém nove cenários de ataque distintos, incluindo reconhecimento, interceção de comunicações, negação de serviço e infeções por botnets, totalizando mais de vinte e sete milhões de instâncias de tráfego. As características foram extraídas com o AfterImage feature extractor, gerando cento e quinze atributos por pacote, capazes de capturar tanto métricas instantâneas como padrões temporais. Para simular a heterogeneidade típica de redes reais, os dados foram distribuídos de forma não uniforme entre os participantes, com proporções variáveis de tráfego benigno e malicioso.

A avaliação experimental incidiu em três dimensões. Primeiramente, a capacidade do modelo global em lidar com distribuições não independentes e não identicamente distribuídas (non-IID), um cenário que reproduz a realidade de ambientes distribuídos. Em segundo lugar, o impacto da personalização no desempenho dos modelos locais, contrastando abordagens puramente globais com versões personalizadas. Em terceiro lugar, a resiliência do sistema face a participantes adversariais, simulando a presença de participantes que enviam atualizações corrompidas ou incoerentes.

Os resultados foram elucidativos. No experimento realizado sob o cenário de dados non-IID, verificou-se que, em configurações com apenas duas classes (normal e um ataque), a precisão alcançada foi elevada, ultrapassando os noventa por cento. Com o aumento do número de classes, a maior complexidade do problema traduziu-se em descidas graduais da precisão, embora a solução tenha mantido valores estáveis próximos dos oitenta por cento em cenários multiclasses, evidenciando a sua versatilidade. A personalização revelou-se determinante, com modelos adaptados superando consistentemente os modelos globais não personalizados. Não apenas a precisão média se elevou, como também se reduziu a dispersão entre desempenhos locais, demonstrando que cada participante conseguiu alinhar o modelo às particularidades do seu tráfego. Em contraste,

sem personalização, alguns participantes registaram quedas de desempenho severas, evidenciando que a abordagem puramente global se revela inadequada em ambientes heterogéneos.

O protocolo de reputação revelou-se eficaz na mitigação de ataques adversariais. Em cenários com participantes maliciosos a introduzir atualizações corrompidas, a ausência de reputação provocou instabilidade na convergência e degradação da precisão. Com o mecanismo ativo, os contributos maliciosos foram rapidamente identificados e penalizados, limitando a sua influência na agregação, o que manteve níveis elevados de precisão e convergência consistente. Métricas como accuracy, precision, recall e F1-score confirmaram estes resultados, evidenciando que a personalização aumenta a precisão local e que a reputação mitiga eficazmente a degradação causada por adversários. Por exemplo, em cenários de 10 classes, a precisão do modelo global permaneceu consistentemente acima de oitenta por cento com personalização, enquanto sem este mecanismo registaram-se quedas acentuadas.

A combinação dos três elementos centrais, aprendizagem federada, personalização e reputação, revelou-se crucial. A aprendizagem federada assegura colaboração entre participantes sem comprometer a privacidade, a personalização adapta o modelo às necessidades individuais e o protocolo de reputação introduz confiança e resiliência, dificultando manipulações adversariais. Esta articulação permitiu construir um enquadramento robusto e escalável, adequado para ambientes distribuídos e dinâmicos, como os das redes multi-organizacionais, distinguindo-se das abordagens existentes ao conjugar descentralização, hierarquia adaptativa e confiança baseada em reputação. O sistema atinge um equilíbrio eficaz entre precisão, robustez e adaptabilidade, respondendo a problemas centrais identificados na literatura: vulnerabilidade a ataques de envenenamento de modelos, degradação em ambientes heterogéneos e limitações de escalabilidade.

O trabalho conclui que a solução apresentada constitui um contributo relevante para a evolução dos IDS baseados em Aprendizagem Federada, ao integrar personalização e reputação numa arquitetura híbrida peer-to-peer capaz de enfrentar desafios de ambientes distribuídos e adversariais. A aplicabilidade em redes heterogéneas e cenários de larga escala é evidente, considerando o volume crescente de dispositivos, a diversidade do tráfego e a ameaça constante de ataques. Trabalhos futuros deverão explorar redes de maior escala, otimizar parâmetros como o número de superpeers e o limiar do mecanismo de reputação, e avaliar cenários adversariais mais sofisticados, incluindo ataques bizantinos e a introdução de novos ataques durante a execução, de forma a consolidar o sistema como uma solução robusta, escalável e resiliente para deteção colaborativa em contextos reais.

Palavras-chave: Sistemas de Deteção de Intrusões, Redes Peer-to-Peer, Aprendizagem Federada Personalizada, Mecanismo de Reputação, Aprendizagem Automática

Abstract

The proliferation of heterogeneous devices has expanded the attack surface of modern networks, exposing the limitations of traditional Intrusion Detection Systems (IDS). Signature-based approaches, while effective against known threats, struggle to detect novel attacks and depend on constant updates. Machine learning has improved detection, but centralised training raises privacy concerns and remains vulnerable to adversarial interference. Federated Learning (FL) addresses these issues by enabling collaborative training without centralising data, yet it faces challenges of heterogeneity, lack of personalisation, and poisoning attacks.

This dissertation proposes a robust, context-aware IDS based on hybrid topology personalised FL. The system combines two phases: an unstructured peer-to-peer phase for decentralisation and scalability, and a structured tree topology where superpeers coordinate aggregation. A reputation mechanism governs superpeer selection, penalising malicious or unreliable peers, while a flooding protocol supports scalable communication. Each peer enhances adaptability by integrating global contributions with personalised local training.

Evaluation on the Kitsune dataset assessed performance under non-IID data, personalisation, and adversarial resistance. Results show that personalisation improved detection accuracy across heterogeneous clients, while the reputation mechanism constrained malicious peers and reduced instability. The system consistently achieved around 80% accuracy in multi-class detection tasks, outperforming non-personalised and reputation-absent configurations.

These findings demonstrate that combining Federated Learning, personalisation, and reputation-based trust management enhances the robustness, scalability, and adaptability of IDS. The proposed framework provides a resilient and privacy-preserving approach to intrusion detection, suitable for deployment in distributed and adversarial real-world network environments.

Keywords: Intrusion Detection Systems, Peer-to-Peer, Personalised Federated Learning, Reputation Mechanism, Machine Learning

Contents

List of Figures

List of Tables

Chapter 1

Introduction

By the end of 2025, IoT Analytics anticipates that there will be around 21.5 billion connected devices, marking a 14% increase from 2024 [66]. This surge in IoT devices reflects the digital world's direction— toward easier, more frequent, and flexible connections. Now, more than ever, people integrate digital connectivity into their lives, fuelling an unprecedented rise in data traffic. Global mobile network data alone, which includes 3G, 4G, and 5G, reached approximately 164 exabytes in 2024, with forecasts suggesting it may rise to 195 exabytes by the end of 2025 [1]. This heightened connectivity has made networks a prime target for hackers, who exploit data's exposure across platforms. As cyberattacks evolve, targeting data confidentiality, availability, and integrity has led to severe financial and operational impacts. In 2023, cybercrime inflicted nearly $8 trillion in costs globally, with projections reaching $10.5 trillion annually by 2025 [22]. Such attacks undermine user trust, disrupt organisational workflows, and impact public administrations.

Devising improved detection mechanisms, particularly Intrusion Detection Systems (IDS), is a core response to these threats, as they play a vital role in identifying potential system or network compromises. Widely used IDS solutions such as Snort, OSSEC, and SolarWinds SEM [58] employ various detection techniques, including signature-based detection, to effectively combat known threats. However, this approach struggles with novel attacks due to the delay between their emergence and their documentation in databases or rule sets for these systems.

Given the evolving nature of cyber threats, Machine Learning (ML) has become a promising solution to enhance IDS [68]. ML's ability to identify patterns and learn from data has significantly advanced IDS capabilities, enabling the detection of previously unknown attacks, facilitating real-time threat identification, and providing greater adaptability compared to traditional, database-dependent systems.

Despite their advantages, ML-based IDS face significant challenges. One key issue is their vulnerability to various attacks, such as class inference attacks, where adversaries attempt to gain access to representative samples of target labels that they do not own; or property inference attacks, that aim to uncover meta-characteristics of other participants' training data [75]. These techniques enable attackers to extract sensitive information, undermining both the security and confidentiality of the ML process and reducing the model's accuracy, which further reinforces concerns about the reliability of some anomaly-based systems when compared to signature-based ones [21].

Moreover, performance limitations arise when handling large datasets, causing slower training times and scalability issues, while insufficient data may hinder model training. The acquisition of large, diverse datasets remains a significant challenge, as many organisations are hesitant to share sensitive information due to privacy concerns. For example, sharing logs or documents about past attacks between organisations could jeopardise their confidentiality and security.

Regarding these challenges, Federated Learning (FL) may serve as a solution. FL is a decentralised training framework in which, over multiple rounds, each client trains the model locally and only sends its parameters to be aggregated by a central server. This ensures that organisations do not need to share their own data, thus preserving data confidentiality during the process. The FL approach can also be complemented with personalisation, known as Personalised Federated Learning (PFL) [42], where each client can adapt back locally the global model to meet its specific needs, improving the client's local accuracy.

The objective of this work is to leverage FL, specifically Personalised Federated Learning (PFL), to develop a hybrid and robust IDS capable of effectively detecting both known and unknown attacks, while also being resilient to attacks targeting the IDS training process.

The system employs a hybrid topology, comprising two phases. The first is an unstructured peer-to-peer (P2P) network, in which each peer is connected randomly. The second is a structured phase, where multiple tree structures are created to initiate the FL process, with servers acting as superpeers and regular peers functioning as clients. The system transitions between these phases, allowing a complete reconfiguration of the network at regular intervals.

Given the inherent data imbalance across clients, each local model is personalised according to the specific needs and context of the client's data. This approach ensures that the global model is continuously refined to reflect the diversity of attack patterns encountered in different environments, while allowing local models to retain their particularities and specialisations based on their own datasets [42].

To further enhance the robustness of the system, we propose a reputation mechanism that assesses each client's contribution to the global model, adjusting reputation levels based on the client's consistency with it. If a client's performance deviates significantly from the expected outcome, the server lowers its reputation level, limiting or even blocking that client's influence on model aggregation. On the other hand, if the contributions are valuable, its reputation increases, making the client trustworthy. Clients with high reputations can be selected as superpeers (servers) and may be deposed if they fail to meet the performance expectations established for the clients under their responsibility. This approach will force the peers to rebuild their connections and re-establish the tree structures they belong to. Such feature helps prevent faulty or malicious clients and servers from undermining the system's performance.

The proposed approach offers several advantages arising from its contributions. By leveraging *Personalised Federated Learning*, it ensures privacy by keeping sensitive data local to each client, eliminates the need for centralised data storage, and enables secure model updates. By combining the collective strength of the global model with the tailored expertise of the local models, the

system optimises performance across a wide range of scenarios, making it more adaptable and efficient in detecting and responding to various threats. Additionally, by employing a *hybrid topology*, the system achieves scalability and flexibility, while reducing training times, as servers manage only a subset of clients rather than the entire network. Finally, the *reputation mechanism* enhances robustness against adversarial participants by limiting their contributions and influence within the system.

1.1 Contributions

The main contributions of this dissertation are the following:

- A comprehensive study of the PFL framework, highlighting its advantages and addressing key challenges such as Non-IID data distributions and scalability issues.
- An analysis of adversarial threats targeting Federated Learning frameworks, including training-targeted attacks, and an exploration of mechanisms to enhance robustness against malicious participants.
- The design and implementation of a hybrid, reputation-based solution that leverages PFL to improve the detection of attacks, with an evaluation of its applicability in real-world scenarios such as intrusion detection systems.

1.2 Document Structure

The organisation of this dissertation is as follows:

- Chapter 2 - Provides the background, defining and explaining the key concepts necessary to understand both the problem at hand and the proposed solution.
- Chapter 3 - Reviews related work, outlining existing solutions, their advantages, and limitations. This review offers insight into the current state of the field and highlights the innovations introduced by the proposed solution.
- Chapter 4 - Presents an overview of the proposed solution, describing the system's workflow, its main components, and how they are interconnected.
- Chapter 5 - Explains how the solution was implemented, detailing the tools and resources used to achieve it.
- Chapter 6 - Evaluates the solution, presenting results that demonstrate the system's performance against the challenges previously described.
- Chapter 7 - Presents a summary of the findings and a discussion of potential directions for future work.

Chapter 2

Background

The Background chapter establishes the essential theoretical foundations necessary to comprehend the various ideas and components underpinning this thesis. It begins with the fundamental concept of intrusion, which forms the basis of this work, and then examines Intrusion Detection Systems (IDS), outlining their classifications, objectives, and key limitations. Thereafter, the concept of Peer-to-Peer (P2P) Networks, integral to the proposed methodology, is presented, highlighting their architectural principles, distinct categories, and the range of strategies adopted within this domain. The chapter concludes with an overview of Machine Learning, with particular focus on the pivotal frameworks central to this research: Federated Learning and its derivative, Personalised Federated Learning.

2.1 Intrusion

An *intrusion* refers to unauthorised activity within a digital network or system, which may constitute a violation or pose a threat to the enforcement of computer security policies, acceptable use policies, or standard security practices [60]. Such intrusions are often driven by malicious intent and may involve unauthorised system access, data manipulation, service disruption, denial-of-service attacks, or the deployment of malicious software. These actions can severely compromise system integrity, leading to data corruption, financial loss, operational interruptions, and reputational damage [77]. However, not all intrusions are explicitly hostile. In certain contexts, it is also possible for an intrusion to occur without malicious intent—for instance, when a user inadvertently gains access to a system, or when access to sensitive information is granted due to a software bug or a failure in the system's security mechanisms.

Intrusions can be classified as either passive or active, based on the nature of the intrusion behaviour. *Passive intrusions* encompass discreet activities in which a threat actor observes, scans, or collects information from a network or system without modifying its functionality or affecting data integrity. These forms of unauthorised access are often subtle and difficult to detect. One typical example is traffic analysis, in which an attacker monitors network transmissions to identify patterns, such as the timing and frequency of packets, without accessing the content itself. This technique remains effective even in encrypted environments, as it exploits communication meta-

data rather than the encrypted payload. Additional examples of passive intrusion methods include keystroke logging and web crawling, both aimed at silently harvesting user input or publicly available data. Eavesdropping involves intercepting data exchanges to capture confidential information, potentially allowing the intruder to decode protected content. In spying, the attacker poses as a legitimate user or endpoint to covertly monitor activity within the system, thereby compromising data privacy under the guise of a trusted connection [71].

Conversely, *active intrusions* involve the direct compromise of digital resources through actions that deny, delete, or restrict access, with the intention of disrupting system behaviour, compromising data integrity, or affecting service availability. A prominent example is the Distributed Denial of Service (DDoS) attack, illustrated in Figure 2.1, where a target system is overwhelmed by a large volume of requests originating from a distributed network of compromised devices [37]. These requests, typically illegitimate, overwhelm the system's capacity, rendering it unable to respond to legitimate traffic. Since the attack originates from numerous sources, blocking a single origin is ineffective, making mitigation particularly challenging. These networks of compromised devices, often referred to as botnets, are typically formed by infecting hosts with malware that allows the attacker to remotely control them. Once compromised, the infected devices can be directed to simultaneously issue requests to a designated target, thereby exhausting its resources.

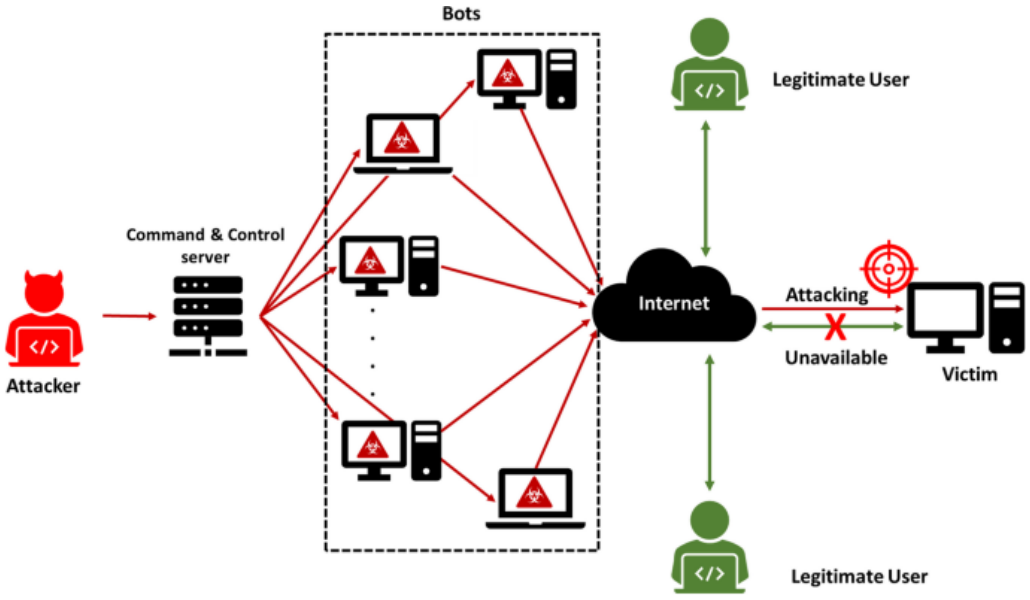

Figure 2.1: Example of DDoS attack using Botnet [16]

Variants of this attack include Yo-Yo attacks [17], which specifically exploit the auto-scaling capabilities of cloud-based systems. In such cases, the attacker initiates a surge in traffic that forces the victim's infrastructure to scale up its resources, incurring increased operational costs. The attack is then temporarily paused or reduced, prompting the system to scale back. Once resource usage returns to normal, the attack resumes, creating a cyclical strain that can result in financial and operational disruption. Another sophisticated variant is the Advanced Persistent DDoS (APDoS) attack, which follows the same basic principles as a traditional DDoS attack but

is distinguished by careful planning, high computational power, and long duration, often lasting for weeks to maintain a sustained impact.

Another form of active intrusion is the Man-in-the-Middle (MITM) attack, wherein the attacker secretly intercepts and potentially alters the communication between two parties. This type of intrusion can compromise confidentiality by exposing private data, availability by discarding or delaying packets and thereby causing data loss, or integrity by modifying transmitted information [63], eroding trust between communicating entities.

Effective strategies to mitigate these threats include a combination of technical measures and user-focused practices. Employee training is essential, particularly in defending against social engineering attacks, which rely on psychological manipulation to deceive individuals into revealing sensitive information or performing unauthorised actions. The deployment of firewalls provides an additional layer of defence by filtering incoming and outgoing traffic based on predefined security rules. Authentication mechanisms play a critical role in access control, as they ensure that only authorised users gain access. Multi-factor authentication improves security by requiring several forms of verification, reducing the risk of unauthorised access, while token-based authentication complements this by issuing a time-limited access token once a user has been successfully verified. Regular data backups are also vital, ensuring data availability and integrity in the event of breaches or ransomware attacks that attempt to encrypt or withhold access to critical data [73]. Finally, the implementation of threat detection systems, such as Intrusion Detection Systems (IDS), allows for the identification and response to suspicious activity within a network or system, thereby enhancing the overall security posture.

2.2 Intrusion Detection Systems

An *Intrusion Detection System (IDS)* is a cybersecurity mechanism designed to monitor and analyse network or system activity for indicators of potential threats or anomalous behaviour that may contravene system configurations or user-defined security policies [36].

Monitoring by an IDS may be either active or passive. In *passive monitoring*, the IDS operates in a non-intrusive manner, observing communication patterns and data flows to identify suspicious actions without directly interfering with system operations. Its principal function is to raise alerts upon the detection of irregularities, which can be delivered via email, text message, or integrated into broader security platforms such as Security Information and Event Management (SIEM) systems [43]. These systems correlate IDS alerts with other data sources to assess whether a security incident is genuine. SIEMs also apply correlation rules to recognise specific attack patterns, such as brute-force login attempts (detected by identifying repeated failed authentication efforts), impossible travel scenarios (where user authentication and access occurs within an implausible time frame), and abnormal file transfer behaviour, such as excessive or unusual copying activity.

In active monitoring, the system—commonly known as an Intrusion Prevention System (IPS)—extends the functionality of traditional detection tools by taking immediate action against identified threats. Unlike an IDS, which passively observes and reports suspicious activity, an IPS

intervenes in real time to mitigate risks [59]. It may block traffic from malicious IP addresses, terminate unauthorised sessions, or alter malicious packets by removing harmful components such as headers or repackaging payloads to prevent delivery of threats. This is achieved through Deep Packet Inspection (DPI), which enables the system to analyse the contents of data packets beyond basic header information, allowing it to detect known malware signatures and other embedded threats. This proactive capability allows an IPS to neutralise attacks while maintaining the flow of legitimate network communication.

IDS are among the most widely adopted methods for detecting attacks in real-world platforms, particularly in sectors such as healthcare, retail, finance, and critical infrastructure. These systems play a vital role in ensuring compliance with various regulations, including the Payment Card Industry Data Security Standard (PCI DSS) [18]. According to PCI DSS Requirement 11.4, organisations are required to use intrusion detection (IDS) techniques to detect or prevent intrusions on the network, thereby establishing security standards for global payment processing.

2.2.1 Types of IDS

IDS are vital for identifying and mitigating security threats in digital environments, where robust and reliable detection mechanisms are essential to counter evolving attack strategies. IDS can be broadly classified into two main categories: signature-based and anomaly-based detection systems.

Signature-based IDS operate by matching incoming data against a predefined database of threat signatures. These signatures represent unique patterns or characteristics of malicious activities, such as fragments of malware code, specific network traffic behaviours, or anomalous user actions. This approach provides high accuracy in detecting attack types that have already been catalogued [5]. Also referred to as knowledge-based detection systems, they are particularly effective against well-known threats, including abuse of functionality attacks (malicious exploitation of legitimate system features), buffer overflow attacks (attempts to overwrite memory regions), and command execution attacks (injection of commands through user inputs to alter system behaviour) [49]. These systems utilise pattern recognition to match observed activity with known signatures and generate alerts upon detection. Widely used implementations of signature-based IDS include Snort, a popular open-source tool [52]; Suricata, known for its high-performance capabilities [51]; and Cisco IDS, which integrates signature databases into its security infrastructure [7].

In contrast, *anomaly-based IDS* focus on detecting deviations from established patterns of normal behaviour, such as logging in at unusual hours or attempting to access files or directories without the necessary permissions. These systems are particularly effective in identifying novel and unknown attack types. Unlike signature-based systems, many anomaly-based IDS, commonly referred to as ML–based IDS, employ machine learning techniques to construct models of typical network behaviour, enabling them to flag significant deviations as suspicious [36]. In IoT environments, anomaly-based IDS can alert administrators to communication anomalies that indicate

emerging threats. This approach is advantageous because it eliminates the need for centralised training data, which can be resource-intensive and raise privacy concerns.

Regarding deployment, *Host-based IDS (HIDS)* monitor activity within individual systems, including user actions and access to log files, to maintain a record of system events that can be analysed for indications of unauthorised or suspicious behaviour [3]. As presented in Figure 2.2, this targeted monitoring provides detailed insights into system-level activity and can detect malicious actions not visible at the network level.

Conversely, *Network-based IDS (NIDS)* monitor network traffic to detect anomalies in communication patterns [47]. Deployed at strategic points within the network, NIDS analyse incoming and outgoing traffic by comparing it against known attack signatures or established anomaly patterns to identify potential intrusions. This approach is particularly effective in end-user networks, as it enables real-time surveillance of device communications, contributing to a more secure network environment. By positioning NIDS at critical network junctions, all traffic within a subnet can be scrutinised. These systems can be further classified based on execution timing: Online NIDS analyse data in real time, allowing immediate detection and response, whereas Offline NIDS store traffic data for later analysis, making decisions about potential intrusions after processing the collected information.

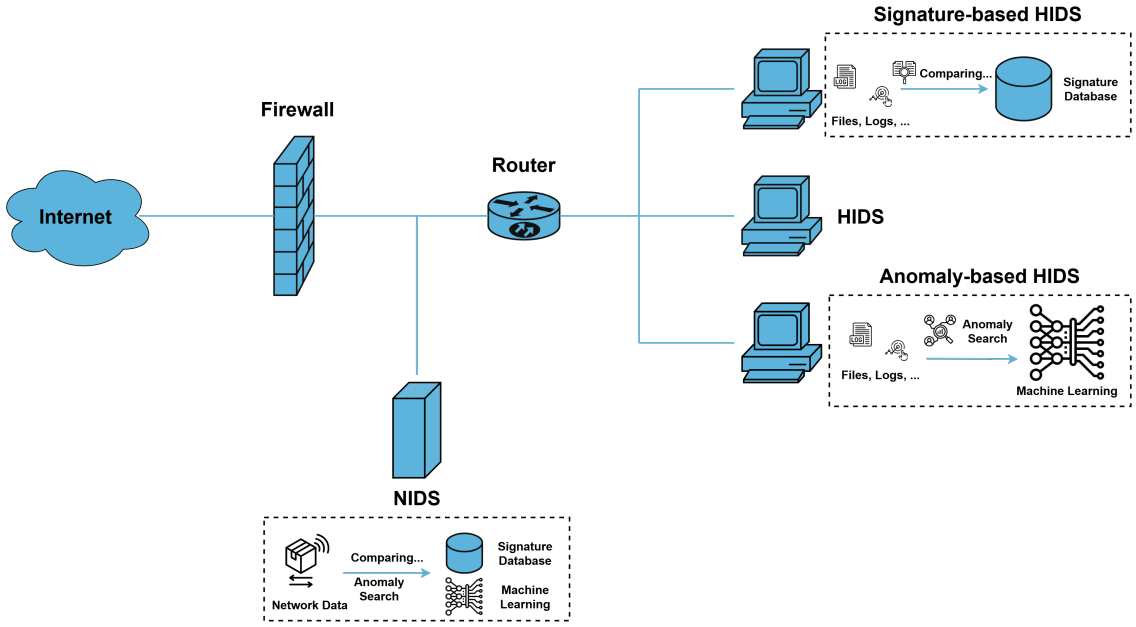

Figure 2.2: Overview of IDS deployment in host and network layers

2.2.2 Limitations

Despite their widespread use, IDS systems face several limitations. One significant challenge is data noise, which can distort or fragment data, undermining the system's effectiveness. For example, packet fragmentation slows down traffic analysis as the IDS must reconstruct fragmented packets into complete sessions, demanding considerable computational resources and time. Ad-

ditionally, the high volume of network or system traffic increases the likelihood of benign content being mistakenly identified as malicious, leading to an elevated rate of false alarms [6]. This results in events being incorrectly classified as intrusions, which diminishes the trust in the system's ability to accurately detect genuine threats.

In many instances, the number of actual attacks is significantly lower than the number of false alarms [33], which can cause real threats to be overlooked. This undermines confidence in the system's effectiveness. Another issue faced by IDS systems is their reliance on signature databases, which, if outdated, can render the system ineffective against new and evolving attack methods. In signature-based IDS, there is often a delay between the identification of a new threat and the inclusion of its corresponding signature into the system. This delay leaves the IDS vulnerable to undetected threats, increasing the risk of a system compromise.

Moreover, IDS systems cannot always compensate for weak identification and authentication mechanisms or vulnerabilities in network protocols. For instance, insecure protocols like HTTP, weak passwords that do not adhere to best practices, or reliance on single-factor authentication can provide attackers with easy access. If an attacker gains access through weak authentication, the IDS may not be able to detect or prevent malicious activity, as it is simpler for an attacker to bypass or evade detection. Additionally, certain IDS are incapable of processing encrypted packets, such as SSL-encrypted web sessions, due to the lack of decryption keys, allowing intrusions to bypass detection. As a result, breaches may go undetected until they escalate into more severe network issues [6].

These challenges can be mitigated by an IDS that analyse and detect network flow patterns, allowing for the identification of suspicious activities, even if they originate from a legitimate user or the packets transmitted are encrypted. Additionally, IDS relies on the accuracy of the network address associated with the IP packets entering the network. If the address is falsified or scrambled, the information collected by the IDS becomes unreliable, rendering it ineffective for implementing mitigation measures, as actions based on incorrect data cannot address the actual threat. NIDS systems, due to their nature, are susceptible to the same protocol-based attacks that can affect network hosts, and invalid data or TCP/IP stack attacks can cause NIDS to crash [62].

2.3 Peer-to-Peer Network

Peer-to-peer (P2P) is a decentralised network model in which each participant, or peer, acts simultaneously as both a client and a server—accessing resources while also sharing them with neighbouring nodes [61]. This stands in contrast to the traditional client-server architecture, where a centralised server controls resource management and distribution.

P2P networks may be classified based on their structural organisation. *Unstructured P2P* networks feature random peer connections with no predetermined topology, making them easier to deploy due to minimal configuration constraints [27]. However, this structural randomness leads to inefficient resource discovery, as there is no deterministic mechanism to locate data across peers. Specifically, when a peer seeks a particular (and rare) data item, the query must be broadcast

throughout the network to maximise the likelihood of reaching a peer that holds the required content. This flooding strategy generates substantial signalling overhead, increases CPU and memory usage—since each peer must process every incoming query—and does not guarantee successful resolution of search requests [64].

Structured P2P networks, by comparison, operate on defined rules for connectivity and data distribution. A prominent example is the Distributed Hash Table (DHT) approach [11] presented in Figure 2.3, which employs a hashing algorithm to organise resources as key-value pairs. In this Figure, supernodes (SNs) are connected in a chord-like circular structure, forming the upper layer of a two-layer hierarchical architecture. Participants are divided into Ordinary Nodes (ONs, named from A to I) and SNs, with the latter elected from ONs based on uptime, processing capacity, bandwidth, and storage. Each SN manages its cluster by maintaining connections with all ONs while also joining the distributed hash table (DHT)-based SN network to support inter-cluster searches. Only SNs maintain up-to-date information on the resources within their clusters. Queries generated by any node are first sent to its local SN: if the target resource is within the cluster, the SN resolves it locally; otherwise, the request is forwarded through the SN overlay. For instance, node E sends a query to its SN: if the target is node C, it is resolved locally; if the target is node F, the query is forwarded through the DHT-based SN network. Once located, a direct connection is established between the requester and the target. This enables peers to efficiently locate values associated with specific keys, such as peer IDs. While *Structured P2P* improves search accuracy and consistency, it also introduces overhead from data advertisement and may create load imbalances due to uneven distribution of resources and requests [11].

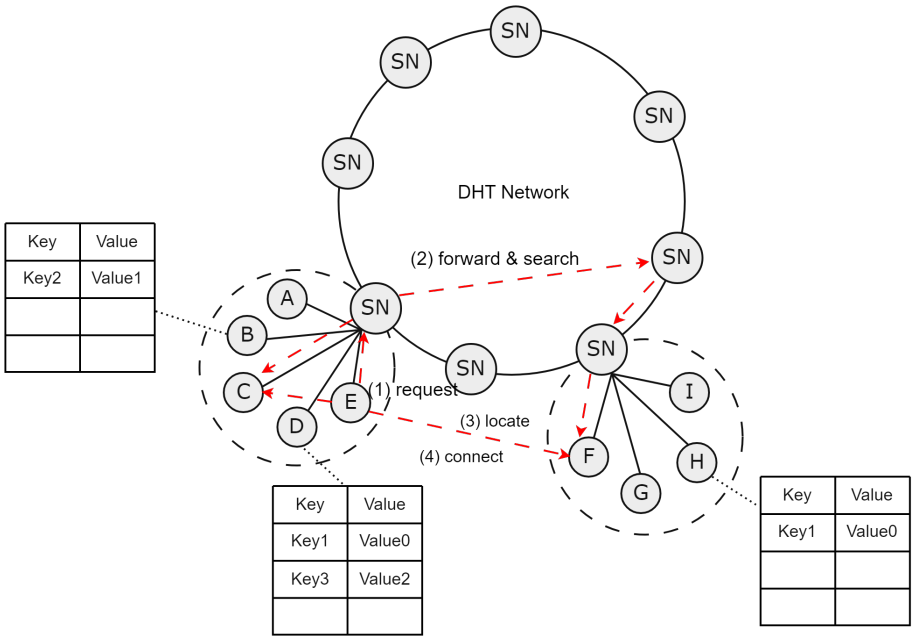

Figure 2.3: Search Model in Hierarchical P2P network

P2P architectures inherently support scalability and fault tolerance. Since there is no centralised control, the failure of individual nodes does not compromise the integrity or functionality of the network. Additionally, P2P systems promote user privacy and anonymity [72]. Communication between peers is often encrypted, and the decentralised nature reduces exposure to surveillance and censorship. These advantages render P2P networks particularly suited to dynamic, distributed applications where robustness, flexibility, and confidentiality are essential.

2.4 Machine Learning

Machine learning (ML), a subfield of artificial intelligence (AI), is concerned with the development of algorithms that enable computational systems to learn from data and improve performance over time without explicit programming [29]. This paradigm, inspired by cognitive learning in humans, enables models to autonomously identify patterns and infer relationships, ultimately allowing them to make informed predictions in previously unseen scenarios.

At the core of this paradigm is the notion of generalisation—the capacity of a model to accurately predict outcomes on new inputs drawn from the same (usually unknown) distribution as the training data [12]. Since this distribution is only indirectly accessible through finite samples, the learning task becomes one of approximation: constructing a hypothesis function that captures meaningful structure while avoiding overfitting to noise.

To formalise learning performance, computational learning theory provides rigorous mathematical tools. The *Probably Approximately Correct (PAC)* learning framework, in particular, offers probabilistic guarantees regarding how well a hypothesis generalises from training data. Central to this is the analysis of the *bias–variance trade-off*, ilustraded it Figure 2.4 which explains how model complexity influences generalisation error. Underfitting results from models with insufficient capacity, while overly complex models may memorise training data and perform poorly on new inputs—overfitting [4].

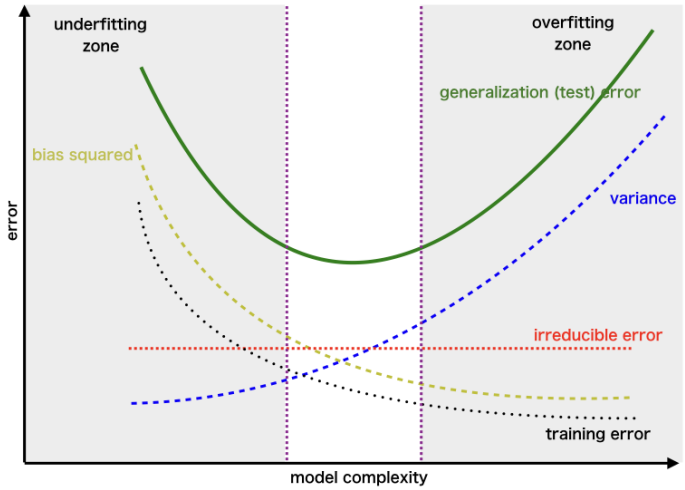

Figure 2.4: Bias and variance as function of model complexity [24]

Beyond predictive accuracy, learning theory also investigates the computational feasibility of training. This includes determining whether a function class can be learned efficiently—typically in polynomial time. Positive results identify function families that are learnable under reasonable assumptions, whereas negative results establish theoretical limits, showing that certain learning tasks are intractable.

The landscape of ML methodologies can be classified according to the nature of training signals. These include supervised, unsupervised, semi-supervised, and reinforcement learning, each addressing different problem structures and data availability constraints.

In *supervised learning*, the model is trained on labelled input–output pairs to approximate a mapping function [56]. This process generally involves optimising a loss function that quantifies prediction error. Tasks such as classification and regression fall within this category. An extension of this approach is similarity learning, where the objective is to learn a function that reflects the similarity or distance between data points, which is particularly relevant in applications like face verification or recommendation systems.

By contrast, *unsupervised learning* involves unlabelled data and seeks to uncover latent structure, such as in clustering or dimensionality reduction [56], as presented in Figure 2.5. A prominent variant is self-supervised learning, which generates supervisory signals internally from the data. This has proven highly effective in pretraining deep models without requiring manual annotation.

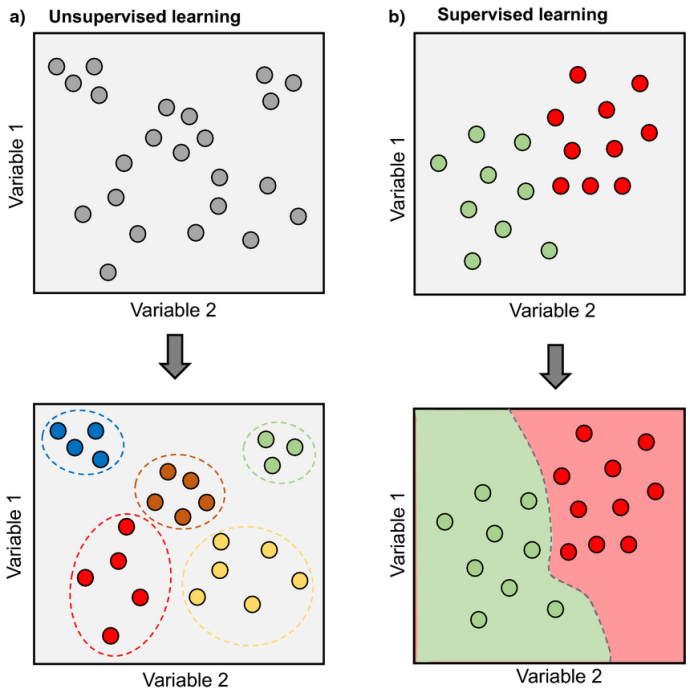

Figure 2.5: Unsupervised and supervised machine learning [45]

Semi-supervised learning combines small amounts of labelled data with large volumes of unlabelled data, leveraging both to improve performance. Relatedly, weak supervision relaxes labelling constraints, allowing models to learn from incomplete or noisy annotations, a valuable

trait in cost-sensitive environments [55].

Reinforcement learning (RL) frames learning as a sequential decision-making process in which an agent interacts with a dynamic environment to maximise cumulative reward. Often formalised via Markov Decision Processes (MDPs), RL is suited to domains requiring long-term strategy, such as robotics or game playing. Unlike supervised learning, RL operates with delayed and often sparse feedback, presenting unique optimisation challenges.

Across all paradigms, the effectiveness of a learning algorithm depends heavily on the choice of model. A machine learning model is a mathematical structure trained to approximate an unknown function. It is typically defined by a set of parameters adjusted through training to minimise a chosen loss function. The term model may refer to a general class (e.g., decision trees) or a fully trained instance.

Among the most prominent models are *Artificial Neural Networks (ANNs)*, which are inspired by biological neurons and consist of interconnected layers of nodes. Each artificial neuron applies a non-linear transformation to a weighted sum of its inputs, and weights are updated during training via gradient descent. Deep learning extends this by stacking multiple hidden layers, enabling hierarchical feature learning. Although highly expressive and successful in domains such as vision and speech, deep models often require substantial data and computational resources.

Where interpretability is critical, decision trees offer a more transparent approach by recursively partitioning the input space based on feature thresholds. They are easy to understand but prone to overfitting. *Random Forests (RFR)* address this by forming an ensemble of decision trees trained on random subsets of data and features, combining their outputs to reduce variance and improve robustness. RFRs are versatile and widely used in structured data tasks.

Support Vector Machines (SVMs) construct hyperplanes that maximise the margin between classes, as illustrated in the case of a linear SVM (Figure 2.6). For non-linearly separable data, the kernel trick allows mapping inputs into higher-dimensional spaces where linear separation is feasible. SVMs offer strong generalisation capabilities, especially in high-dimensional contexts, though their scalability can be limited on large datasets.

Model selection thus involves navigating trade-offs: neural networks provide flexibility but can be opaque; decision trees are interpretable but unstable; ensemble methods like random forests balance accuracy and robustness; and SVMs offer reliable performance under specific data conditions. These considerations are particularly relevant in sensitive domains such as security, medicine, or autonomous systems, where performance, explainability, and resource constraints must all be weighed.

2.5 Federated Learning

Federated Learning (FL) is a decentralised machine learning framework that prioritises data privacy by keeping training data local to each client and only transmitting model updates to a central server [34]. Typically structured within a client–server architecture (see Figure 2.7), FL involves a central server that aggregates model parameters shared by distributed clients [3]. The process

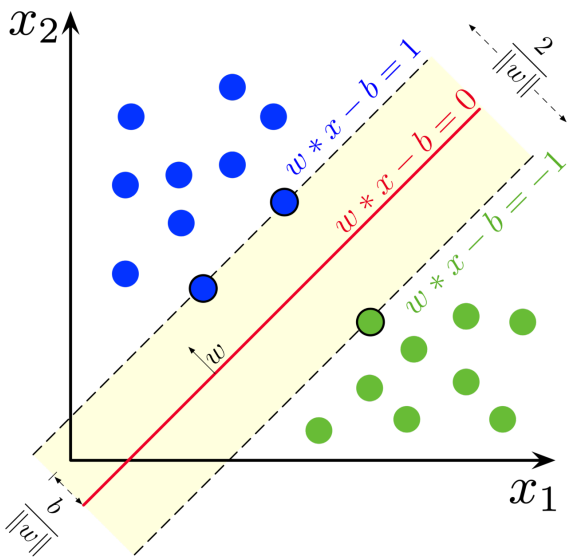

Figure 2.6: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes

begins with the server initialising the model parameters and distributing them to the participating clients, each with distinct and heterogeneous local datasets. Clients then train the model on their respective datasets, adjusting parameters to enhance accuracy. These updated parameters are sent back to the server, where they are aggregated to construct a global model. This iterative process, known as a *round*, is repeated until the global model meets predefined criteria [3].

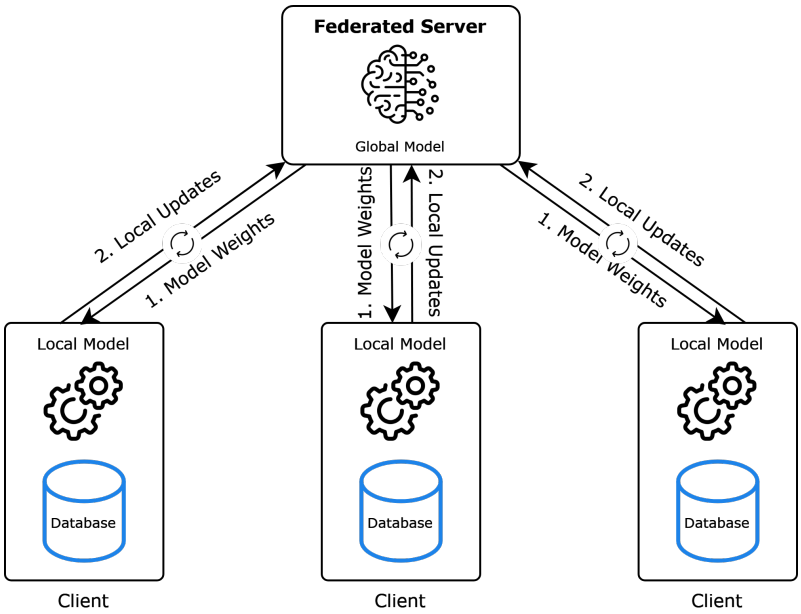

Figure 2.7: Federated Learning

FL can be classified based on various factors, such as data partitioning, the machine learning model used, privacy mechanisms, communication architecture, and the scale of the federation. For example, in terms of data partitioning, FL can operate with horizontal data, where clients

have datasets with the same features but minimal sample overlap; vertical data, where datasets share samples but differ in features; or hybrid partitioning, combining both horizontal and vertical configurations. The scale of the federation [38] also affects FL approaches: cross-silo FL involves organisations with substantial computational resources and storage, while cross-device FL relies on numerous devices with limited computational capacity, like smartphones. The choice of FL type depends on the environment, devices, and use case. For instance, Google Gboard uses both horizontal data partitioning and cross-device FL to implement features like next-word prediction and autocomplete efficiently [25]. On the other hand, vertical data partitioning is often applied in collaborative scenarios between organisations or individual users, such as technology firms sharing distinct but complementary datasets.

In terms of scalability, cross-silo FL is commonly employed in fields like healthcare, for diagnostic predictions, and cybersecurity, for detecting new intrusions, enabling secure data collaboration across multiple entities [38]. FL enhances privacy by decentralising data processing, making it especially suitable for Internet of Things (IoT) networks, where data sensitivity is crucial. Furthermore, it improves training efficiency by distributing the workload across multiple clients, which supports scalability and reduces overall training time.

FL presents several statistical and operational challenges that hinder its effectiveness in real-world deployments. A central issue is data heterogeneity: local datasets vary not only in size and feature distribution but also in quality and underlying biases, often deviating significantly from the global data distribution. These discrepancies, combined with temporal heterogeneity, in which local data distributions shift over time, hinder model convergence and reduce generalisation. Interoperability further complicates training, as data schemas, feature spaces, and annotation standards may differ across clients. Moreover, local datasets often require ongoing curation to remain relevant, yet FL's privacy-preserving constraints limit central oversight. The absence of global visibility into training data makes it difficult to detect and mitigate hidden biases or fairness issues related to sensitive attributes. Privacy constraints also open the door to backdoor attacks, where adversarial clients inject poisoned updates that compromise the global model. The system's robustness is further challenged by partial or total loss of updates due to client failures or communication issues. In addition, many clients lack sufficient labelled data for supervised learning, and variations in hardware, network bandwidth, and processing capabilities introduce further complexity to synchronous and asynchronous aggregation strategies. These factors collectively demand adaptive, robust, and secure FL frameworks capable of maintaining performance under highly dynamic, non-IID, and resource-constrained conditions.

2.6 Personalised Federated Learning

Personalised Federated Learning (PFL) effectively addresses the challenge of data heterogeneity among clients by customising model training to enhance performance in diverse scenarios, where data characteristics vary significantly across participants [41]. This approach is particularly beneficial in fields like healthcare or IoT environments, where differences in participant preferences,

data collection methods, and the intrinsic attributes of the participants themselves contribute to data variability.

Several methods within the PFL framework facilitate this personalisation. One approach is *local training*, where each client independently trains a model on its own dataset without collaborating with others [65]. Once training is complete, the client sends its local model to the central server, which aggregates them to construct a global model and redistributes it back to the clients for further refinement. This method is particularly effective when clients possess substantial local data, providing a strong foundation for FL algorithms [53].

Another strategy is *Multi-Task Learning (MTL)*, which leverages task relationships across clients to create individualised models by addressing multiple tasks simultaneously [19]. In the context of FL, MTL frames both the training and personalisation processes as related tasks, enabling the exploitation of their inherent connections. The strength of MTL lies in its regularisation mechanism, which differs from conventional regularisation techniques. Rather than penalising all forms of complexity uniformly to avoid overfitting, MTL uses task-specific insights to enable more targeted and effective learning.

Additionally, *fine-tuning* presents a model-agnostic solution for personalisation in PFL, especially in stateless scenarios, where clients do not need to maintain relationships with other clients, as is required in MTL. In this method, a global model is initially trained across clients, and then each client performs local fine-tuning based on its own data, resulting in a customised model optimised for inference. By focusing on client-specific requirements, PFL enhances performance for individual clients, leading to better test outcomes that are more aligned with each client's unique data distribution.

Chapter 3

Related Work

Federated Learning (FL) has emerged as a promising paradigm to enable collaborative model training across multiple distributed clients while preserving data privacy. In the context of Intrusion Detection Systems (IDS), FL allows various edge devices, such as IoT nodes or network endpoints, to train shared models on decentralised traffic data without exposing sensitive information. However, the deployment of Personalised Federated Learning-based Intrusion Detection Systems (PFL-IDS) faces several challenges, including non-independent and identically distributed (*non-IID*) data, personalisation requirements, adversarial threats such as poisoning attacks, trust and accountability in aggregation, and the need for scalable, efficient, and explainable systems. A few approaches have been proposed to address these issues, often targeting multiple aspects simultaneously.

3.1 Non-independent and identically distributed (non-IID) data

One of the most fundamental challenges in FL-IDS is the presence of *non-IID* data across clients, which refers to differences in the data distributions due to network heterogeneity, device behaviour, or local traffic profiles. The standard aggregation method, *Federated Averaging (FedAvg)* [38], assumes homogeneity in client contributions and sometimes fails to converge or yields suboptimal global models in such conditions. To address this, several methods have focused on personalisation, aiming to provide models tailored to individual clients while still benefiting from collaborative training.

A foundational approach in this direction is *Ditto* [39], which formulates FL personalisation as a bi-level optimisation problem. Each client learns a personalised model locally while participating in the training of a global model. By adding a regularisation term that penalises divergence from the global model, *Ditto* enables a trade-off between generalisation and personalisation, allowing clients to retain local specificity without entirely disregarding the global consensus. Similarly, *FedDef* [15] introduces a two-stage approach where clients fine-tune a received global model on their local data to improve performance under *non-IID* conditions. This strategy reduces the risk of catastrophic forgetting and allows each client to adapt the global model to its specific data distribution.

Another strategy to improve adaptability under heterogeneity is *FedAdapt* [74], which applies meta-learning principles to FL. By leveraging Model-Agnostic Meta-Learning (MAML), clients are trained to quickly adapt to new tasks or data distributions with minimal fine-tuning. This enhances both convergence speed and model generalisability, particularly beneficial in dynamic environments where client behaviour evolves over time. *FedAdapt* also introduces techniques to reduce communication overhead by identifying and training only on the most informative model parameters, addressing scalability concerns alongside heterogeneity.

3.2 Personalisation

Beyond personalisation via fine-tuning or regularisation, alternative approaches have sought to redefine aggregation mechanisms themselves. For example, *FedFomo* [78] (Federated Follow the Most Personalised) replaces the global aggregation step with a dynamic, client-specific aggregation. Each client evaluates received models from other clients and weighs their influence based on observed historical performance. This approach allows flexible participation and better performance in environments where clients differ significantly. Similarly, *FedAMP* (Federated Attentive Message Passing) [30] employs an attention mechanism to adjust the weight of each client's update based on its relevance to others. Rather than enforcing a single global model, *FedAMP* facilitates personalised models that incorporate knowledge selectively from similar peers, improving convergence and performance under data heterogeneity.

A related method, *FedProto* [69], builds on the concept of prototype-based learning, where each client computes class-wise feature prototypes and shares them rather than raw gradients or model parameters. These prototypes are aggregated to build a global prototype representation, which clients can use for classification tasks. This approach reduces communication cost and enhances privacy but can suffer from decreased accuracy when clients exhibit large inter-class or intra-class variance, limiting its applicability in highly heterogeneous settings.

3.3 Robustness

Another key concern in FL is its susceptibility to attacks, specifically poisoning attacks. An example is MIGO (MIngle with the GOod) [46], a backdoor strategy that generates malicious updates indistinguishable from benign ones. By constraining updates within controlled parameter regions, namely the Effective Search Region (ESR) and the Model Projection Region (MPR), MIGO introduces backdoors gradually, avoiding detection while preserving task utility. It leverages mixed datasets of clean and poisoned samples to maintain consistency with legitimate clients, and supports in-distribution, edge-distribution, and out-of-distribution backdoors.

Early works have demonstrated that FL systems using naive aggregation such as *FedAvg* are susceptible to such attacks due to the lack of robustness in update aggregation [38]. To address this, a range of robust aggregation techniques has been proposed. *FLACI* [36] incorporates trust management by assigning trust scores to clients based on their historical contributions and

observed behaviour. It combines this trust framework with robust aggregation rules and local validation mechanisms, effectively reducing the influence of malicious clients while maintaining participation diversity. Related efforts include *Krum* [79] and *Bulyan* [35], which filter updates by selecting only those that are most similar to others, based on distance metrics in parameter space. While these methods increase robustness against outliers, they often come with increased computational complexity and may inadvertently suppress valuable updates from benign but diverse clients.

Further analysis by Tolpegin et al. reveals the structural vulnerabilities of FL under poisoning attacks, demonstrating that even a small number of attackers can significantly influence the global model [70]. These insights have motivated the integration of *Generative Adversarial Networks (GANs)* into FL-IDS frameworks. In *FL-GAN* [76], clients use GANs to generate synthetic traffic data that enhance model diversity and robustness. By creating realistic but artificial samples, GANs can help detect underrepresented patterns and mitigate class imbalance, which is often exploited in targeted poisoning. Similarly, *FEDGAN-IDS* [67] employs local GANs to augment data before federated training, increasing resilience to model degradation from skewed data or adversarial manipulation.

Trust and accountability in FL systems are further enhanced through *trust management frameworks* and *blockchain integration*. *FLTrust* [14] proposes a central root client with verified data to validate and normalise received updates. This mechanism ensures that malicious updates deviate minimally from a trusted reference. However, it introduces a central point of trust, potentially compromising the decentralised ethos of FL. To overcome this, blockchain-based frameworks such as *FELICIA* [54] and *BlockFL* [32] have been proposed. These systems leverage blockchain technology to record training updates and client actions in immutable ledgers, facilitating traceability, auditability, and accountability. While effective in ensuring transparency, the added blockchain layer increases latency and computational overhead, raising concerns regarding scalability and responsiveness in real-time IDS scenarios.

3.4 System Design Challenges: Scalability, Efficiency, and Explainability

Scalability and communication efficiency also play a critical role in the practicality of FL-IDS. Approaches such as *FedProx* [40] attempt to mitigate the instability of *FedAvg* under *non-IID* settings by introducing a proximal term that restricts the distance between local and global models during training. This regularisation improves convergence and reduces divergence between clients without altering the communication protocol significantly. In contrast, personalisation-based methods like *FedFomo* or *Ditto* indirectly reduce communication by relaxing the need for strict synchronisation, allowing clients to focus on local relevance rather than global consensus.

Aditionally, *explainability* in FL-IDS remains an emerging but underexplored area. The lack of transparency in deep learning models, particularly under federated settings, limits their adoption in domains requiring accountability, such as critical infrastructure or healthcare. Preliminary work

such as *XAI-FL* [20] introduces model interpretability using SHAP (SHapley Additive exPlanations) values, enabling clients to understand and justify predictions made by federated models. This direction is essential for aligning FL-IDS with regulatory and operational needs, but further integration of explainability into the broader FL framework remains limited.

3.5 Final Remarks

In summary, the landscape of FL-IDS research reflects a concerted effort to address the interrelated challenges of heterogeneity, robustness, scalability, communication, and performance. Personalisation strategies such as *Ditto*, *FedFomo*, and *FedAMP* respond to the limitations of global aggregation in *non-IID* contexts. Robust aggregation rules and adversarial defences, including *Robust-FL*, *Krum*, and GAN-based augmentation, strengthen the system against poisoning. Trust frameworks and blockchain integration promote accountability, while communication-efficient designs like *FedProx* and prototype sharing improve scalability.

However, several limitations can be found in the existing literature. Most FL solutions adopt the server–client architecture, which restricts flexibility. In many cases, the performance of FL training is constrained by the number of clients a server can manage, leading to workarounds such as reducing communication, as in *FedFomo* and *Ditto*, or training only selected model parameters, as in *FedAdapt*, in order to address this performance limitation and improve scalability. Another major limitation is the frequent absence of trust mechanisms, as shown in Table 3.1. Several of the proposed solutions do not consider adversarial participants that may take part in the system, leaving a significant gap for research. When resilience is addressed, further limitations may arise due to reliance on a central unit, as in *FLTrust*, where all trust validation is concentrated in a single point, increasing overhead and computational complexity. In other cases, trust mechanisms may be overly rigid, excluding clients with benign yet outlier contributions.

Table 3.1: Comparison of System Features

Proposal	**Personalisation**	**Robustness Mechanism**	**System Efficiency Methods**
FedAvg	×	×	×
Ditto	✓	✓	✓
FedDef	×	✓	✓
FedAdapt	×	×	✓
FedFomo	✓	×	✓
FedAMP	✓	×	✓
FedProto	✓	×	✓
FLACI	×	✓	×
Krum	×	✓	×
Bulyan	×	✓	✓
FL-GAN	×	✓	✓
FEDGAN-IDS	×	✓	×
FLTrust	×	✓	×
FELICIA	×	×	✓
BlockFL	×	✓	✓
FedProx	✓	×	✓
XAI-FL	×	×	✓

Chapter 4

Design

This chapter presents the system design of the proposed robust, reputation-based hybrid personalised Federated Learning (FL) framework. It outlines the overall architecture, emphasising the constituent modules and the interactions that enable collaborative learning.

The chapter begins by establishing the context, outlining the objectives of the framework and the challenges it seeks to address. It then presents a top-level overview of the system's operation, defining its main elements (Peers), primary phases, and overall workflow. Subsequently, the chapter examines the two core phases of the framework and the associated network topologies that support the learning process. Each module is analysed individually, with emphasis on its internal mechanisms, specific responsibilities, and contribution to system reliability and performance. Finally, the protocols that support the framework, namely reputation mechanism, superpeer selection, and flooding-based communication, are described and examined in detail.

4.1 Context

As discussed in Chapter 3, research on FL-based intrusion detection systems continues to face a number of open challenges, which often stem from the inherent complexity of decentralised environments. A major difficulty lies in the presence of non-IID data distributions, where the diversity of local datasets reduces the effectiveness of conventional aggregation strategies. While such strategies aim to construct a centralised global model, this is frequently achieved at the expense of local model performance, limiting the adaptability of the system to individual participants. This issue becomes particularly critical when combined with the threat of adversarial behaviour: the absence of robust mechanisms to detect or mitigate the impact of malicious or unreliable nodes can compromise the integrity of the overall model.

Alongside these concerns, scalability and efficiency also remain pressing limitations. Many existing solutions rely on rigid topologies or communication protocols that are unable to cope with the scale and dynamism of modern networks, often overloading central aggregators and hindering deployment in real-world scenarios.

Variable	Description
$netr$	Network Round
r	FL Round
p	Peer
sp	Superpeer
R	Number of Rounds
$\mathbb{R}\$$	Set of Rounds
$\mathbb{P}$	Set of Peers
$\$\mathbb{P}$	Set of Superpeers
$\mathbb{N}\$$	Set of Neighbours of each peer
θ	Minimum Reputation Threshold
W	Federated Learning Model
MAX_CHILDS	Maximum childs per superpeer

Table 4.1: Algorithm variables and their descriptions

4.2 Architecture

Following the limitations discussed in Chapter 3, the approach proposed in this dissertation introduces a robust hybrid system that integrates reputation mechanisms into a personalised FL-based intrusion detection framework. The design is conceived to address three major challenges identified in the literature: the presence of non-IID data distributions, the need for scalability in large and dynamic networks, and the resilience against adversarial or unreliable participants.

The system is organised as a bidirectionally connected peer network, where each peer connects to a random subset of other peers (denoted as $\mathbb{N}\$$ in Table 4.1), forming its neighbourhood. The process begins in the *P2P Phase*, during which the network remains unstructured. At this stage, all peers start at the initial rounds ($netr = 1$, $r = 1$), share an identical default reputation, and initialise their local models independently. Each peer assigns its vote to the neighbouring peer, or to itself, with the highest reputation score, with the vote being disseminated throughout the network. After the voting procedure concludes, that is, when all peers have voted and the results are known to the entire network, the peers with the highest number of votes are elected as superpeers, and the system transitions to the *Structured Phase*.

In the *Structured Phase*, the FL process is initiated. Each superpeer assumes the role of root within a tree topology, coordinating its subset of peers. Training proceeds iteratively, with model performance evaluations conducted at each round through the reputation mechanism, which monitors the behaviour of both peers and superpeers. Upon detection of a faulty peer, its contribution

is down-weighted, whereas a malicious or unreliable superpeer may be demoted, triggering a re-structuring of the network topology. If no superpeer is identified as faulty after R rounds, the system transitions back to the *P2P Phase*.

The cyclical strategy, illustrated in Figure 4.1, aims to balance decentralisation and structure, mitigating bias while promoting the generalisation of the global model across all peers in the network. The system can leverage the flexibility, distribution, and scalability of P2P networks, enabling natural connections, which is particularly important in real-world deployments where participant numbers and data volumes can vary significantly. At the same time, it benefits from the efficiency of a structured network when transitioning to a tree topology, as the number of communications required to propagate and aggregate models is considerably lower in this client-server configuration than in a fully distributed P2P network.

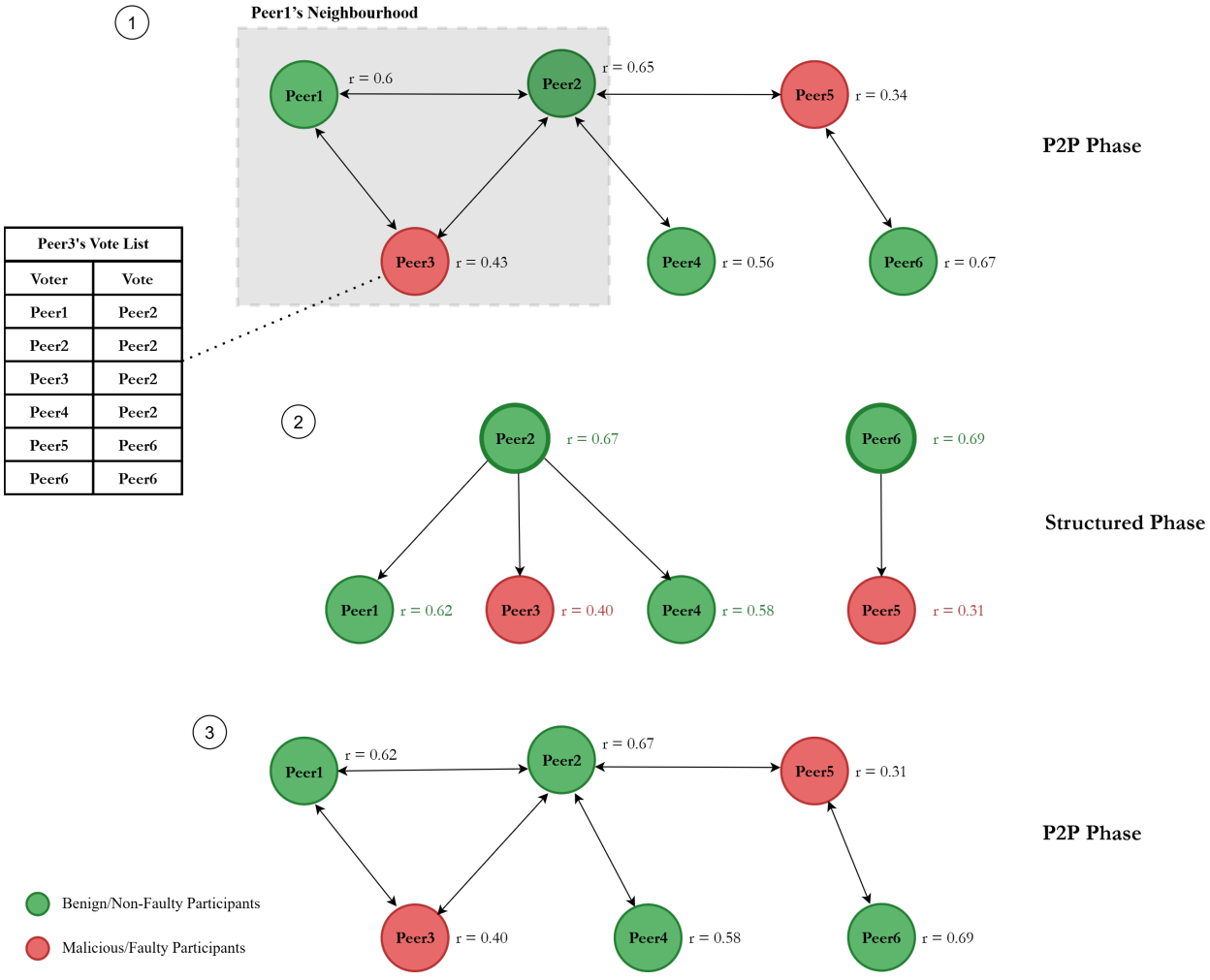

Figure 4.1: Transitions Cycle-Based Solutions Architecture

4.2.1 Peer

The peer constitutes the fundamental unit of the system, representing any organisation that possesses data for training and sufficient computational resources to execute the operations defined in the proposed approach. As described in Section 4.2, the system consists on a network of peers, where the interactions and relationships among them form the core value of the solution.

Within this network, a peer can assume one of two roles. A regular peer has no additional responsibilities and acts as a client in the FL process, solely performing local training and contributing updates to the global model. In contrast, a superpeer functions as a server in the FL process, managing the peers under its responsibility and aggregating their models to produce the global model. Superpeers are elected through the Superpeer Election Protocol (detailed in Subsection 4.3.2), and their responsibilities begin during the Structured Phase.

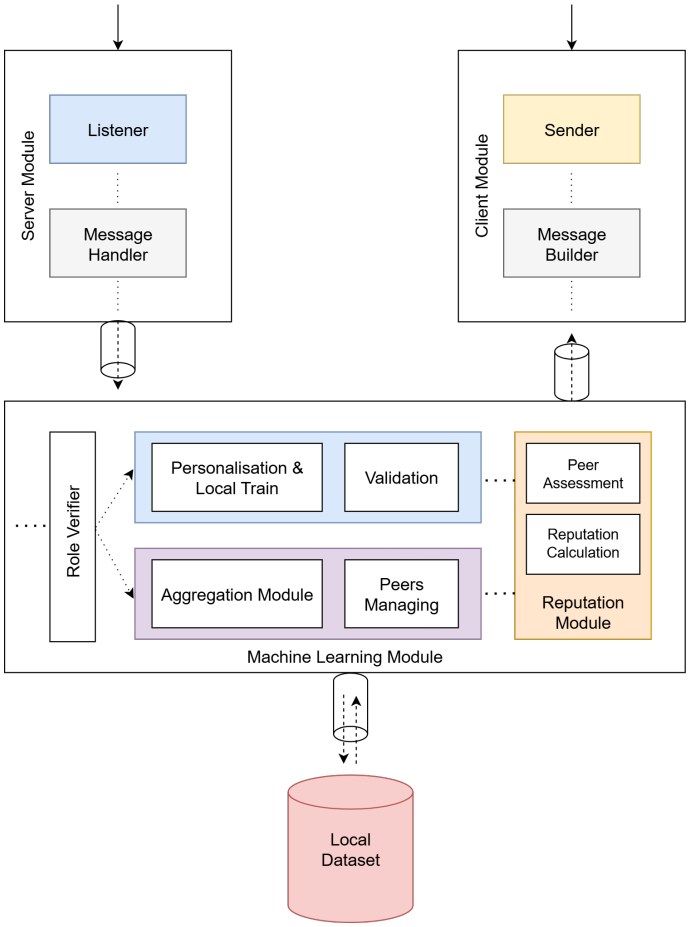

Figure 4.2: Peer's Architecture

Each peer is composed of two communication-oriented components: the server side and the client side, as illustrated in Figure 4.2. The server side handles incoming communications and requests from other peers, including exchanging votes, information, or model parameters. All requests follow a predefined structure to ensure a consistent and reliable communication protocol. Conversely, the client side is responsible for initiating all outgoing connections. It constructs

and sends all messages, as well as initiates actions. Both components operate in parallel and are critical, as effective communication is essential for the operation of this hybrid, distributed system.

Beyond the communication modules, the Machine Learning Module plays a central role in the implementation of the FL approach. This module performs all ML and reputation-related operations, interacting with the communication modules to operate seamlessly in the background and accessing the peer's local data directly. Once an operation is completed, the results are transmitted through the communication modules to their destination.

The behaviour of this module depends on the role of the peer. For a regular peer, the operations are limited to training, validation, and personalisation, which represent the standard responsibilities of a client in the FL process. When a peer is promoted to a superpeer, the module not only performs training but also handles aggregation operations, reputation calculations, and the management of all peers within its tree. The operations are thus implemented in an ad-hoc manner according to the peer's role, ensuring that each peer contributes appropriately to the network.

4.2.2 P2P Phase

The *P2P phase* defines the system's initial topology and reflects its unstructured nature, as outlined in Section 2.3. At this stage, each peer is randomly connected to a number of neighbours, determined a priori by the system configuration (detailed in Subsection 5.1.1). While the number of connections may vary among peers, an upper bound is enforced to prevent excessive connectivity, ensuring balanced communication loads and avoiding bottlenecks. This approach preserves the typical advantages of unstructured P2P systems, including decentralisation, fault tolerance, and scalability.

Each peer may have as few as one or up to the maximum permitted number of connections, resulting in a flexible yet bounded network structure. All connections are bidirectional, facilitating symmetric communication and mutual awareness. However, this topology presents challenges in terms of efficient peer discovery and message dissemination. To address these, the system employs flooding-based protocols (see Subsection 4.3.3), where peers periodically exchange partial network views. Although flooding enables robust and scalable communication, it introduces drawbacks such as increased signalling traffic and probabilistic delivery guarantees, as previously discussed.

Initially, all peers operate with identical roles and reputational status. No distinction exists between nodes in terms of trust or responsibility, as no model training or evaluation has occurred to inform reputation values. Consequently, each peer is initially considered a basic unit, engaged only with its direct neighbours.

This unstructured configuration constitutes the system's default state. As training progresses, the system transitions into the *Structured phase*, where peer roles and topologies become increasingly organised to optimise communication and reduce transmission overhead. Upon completion of the *Structured phase*, or in response to specific events such as a drop in a superpeer's reputation below a critical threshold, the system reverts to the unstructured P2P configuration. However, this

return is not entirely naive: reputational information from earlier phases is retained. As a result, peers may no longer be equally eligible to assume elevated roles, such as superpeers, which are assigned based on the updated reputation scores.

4.2.3 Structured Phase

The *Structured phase* initiates when the network adopts a tree topology, with superpeers serving as roots and being interconnected with one another. This topology is designed to align with the ideal structural environment and support the FL process by reducing communication complexity, as completing a FL round requires fewer communications in a limited client-server tree topology than in a fully broadcasted P2P network.

The tree topology, illustrated in Algorithm 1, is established through the superpeer selection process (detailed in Subsection 4.3.2), during which peers organise themselves by connecting to the selected superpeers. To maintain balance, peers are expected to distribute themselves across all superpeers, ensuring that each tree contains a similar number of peers. This is enforced by assigning a predefined maximum number of peers per superpeer (denoted as MAX_CHILDS), preventing any superpeer from exceeding its allowed capacity.

Once all peers are assigned to their respective superpeers, the *Structured Phase* begins, and the FL process is consequently initiated. If the network round, denoted as $netr$[1], is the initial one, i.e., $netr = 1$, the models of the superpeers are broadcast to their child peers, marking the start of the FL process rounds (step 5). Otherwise, the peers retain the models from the previous network round, preserving the progress achieved so far (step 7).

The *Structured phase* persists until one of two conditions is met. The first occurs when a superpeer is detected as faulty, that is, when its reputation falls below a defined minimum threshold represented by θ. The second occurs when a predefined number of rounds, as specified in the system configuration, has been completed (step 8). In the case of a faulty superpeer, the tree rooted at that node is dissolved, and its associated peers revert to the unstructured P2P topology (step 19). These peers are thereby disconnected from their former superpeer, indicating the necessity of reconstructing the tree. During this period, each peer retains the last known state of the model until a new superpeer is elected. Once the dissolved tree is restructured, the affected peers transmit the preserved model state to the newly elected superpeer, resuming the process from the most recent consistent state and ensuring both continuity and integrity of the model.

Alternatively, if the FL process completes successfully, with all predefined rounds executed and the superpeer exhibiting no faulty behaviour, the existing tree structures are dissolved and new ones are established through the superpeer election protocol. This reorganisation is intended to prevent potential biases in the training process and to ensure fairness, impartiality, and diversity in the contributions made to the FL system.

Once the network returns to the unstructured phase, as previously noted, reputation values are

[1]Network Round – the round defined when the network completes a full cycle of transitions, i.e., when the system moves from the P2P phase to the Structured Phase and back to the P2P phase

Algorithm 1 Structured Phase

1: **Tree Topology:**
2: **for** $sp \in \mathbb{SP}$ **do**
3: **for** $p \in \mathbb{P}$ **do**:
4: **if** $netr \neq 1$ **then**
5: Peer p drops W_0 and adopts superpeer sp's model;
6: **else**
7: Peer p keeps its local model W_l;
end
8: **while** $r < R \wedge sp\neg$ faulty **do**
9: Superpeer sp aggregates the models of all its child peers p;
10: Update reputation of each peer;
11: **if** Peer's Reputation $rep_p \leq \theta$ **then**
12: Remove peer until the end of $rs \in \mathbb{RS}$;
13: Return model W^r to peers;
14: Each peer p evaluates W^r;
15: Each peer update superpeer sp reputation;
16: **if** Superpeer's Reputation $rep_{sp} \leq \theta$ **then**;
17: Peer p broadcasts an ML phase end message and
18: All peers $p \in sp_p$ disconnects from superpeer sp;
19: **return** to P2P topology;
20: Each peer personalises W_r and saves it as W_l;
end
21: **return** Federated Learning Model W^G.
end

preserved across all peers. This allows the system to maintain knowledge regarding which peers are more reputable or trustworthy than others.

The global model is obtained through the aggregation of peers local models within each tree. Superpeers are responsible for collecting all contributions from the peers in their respective trees (step 9). Each contribution is evaluated through the reputation mechanism, assigning a reputation value to the contributing peer and propagating it across the tree. If a superpeer detects that a peer's reputation falls below the minimum threshold, that peer is excluded from the FL process (step 12). Conversely, if peers identify that a superpeer exhibits faulty behaviour, they initiate a deposing process and dissolve the tree to which they belong.

4.3 Protocols

The proposed solution includes three essential protocols that ensure robustness against faulty or malicious clients while supporting the hybrid, complex, and adaptive nature of the system. These are the Reputation Protocol, the Superpeer Selection Protocol, and the Flooding Protocol.

4.3.1 Reputation Protocol

The reputation mechanism is fundamental in assessing each peer's performance and assigning a corresponding reputation level. This reputation directly influences a peer's impact on the FL pro-

cess, with higher reputation levels granting greater weight to the peer's contributions in updating the global model. Peers whose reputation falls below a predefined threshold risk exclusion from the FL process until its completion, ensuring that only reliable participants affect the model.

Beyond influencing contributions, the reputation mechanism also governs the system's hierarchical structure by determining which peers are promoted to or demoted from superpeer positions. Superpeers are evaluated based on performance: if the global model's accuracy lags behind the accuracy achieved by the peers they manage, they incur penalties that reduce their reputation. This evaluation occurs each time the global model is shared, with peers assessing it against their local datasets and updating the superpeer's reputation accordingly (see Equation 4.1). Conversely, consistently high model accuracy leads to reputation gains, increasing the peer's influence and responsibilities within the network. This dynamic creates a feedback loop where peer performance directly shapes both reputation and role within the system.

The reputation of each peer is calculated according to the formula

$$Rep_t = \alpha \cdot Rep_{t-1} + (1 - \alpha) \cdot acc_t \tag{4.1}$$

where:

Rep_t = Reputation of the peer at time instant t

α = Smoothing factor, with $0 \leq \alpha \leq 1$

acc_t = Accuracy of the peer's model at time instant t

The formula ensures that the reputation value always remains between 0 and 1, where values closer to 0 indicate poor reputation and values closer to 1 indicate better reputation. This formula updates the reputation based on both the previous reputation and the latest accuracy obtained from the most recent model evaluation. By incorporating the previous reputation, the system allows peers to be continuously evaluated, reflecting their historical contributions over time.

A smoothing factor, denoted as α, controls the relative influence of past reputation versus recent performance. Specifically:

- When α is close to 1, the update places greater emphasis on the past reputation, causing the system to react slowly to recent changes in performance.
- When α is close to 0, the update prioritises the most recent accuracy, making the reputation more sensitive and responsive, but potentially more volatile.

This value can be adjusted prior to execution, depending on the type of behaviour desired from the system.

4.3.2 Superpeer Selection Protocol

The selection of superpeers is a critical component in implementing the hybrid approach within this system, as superpeers serve as the roots of the trees during the structured phase. The maximum number of children per superpeer (MAX_CHILDS) and the number of superpeers ($\mathbb{SP}$) are

determined based on the desired network structure. A higher number of superpeers results in more trees, each containing fewer peers, whereas a smaller number of superpeers leads to fewer trees with larger peer groups. The choice of $\mathbb{SP}$ involves a trade-off between the benefits of larger trees, such as improved aggregation capabilities, and smaller trees, which reduce overhead and increase adaptability.

Algorithm 2 illustrates the behaviour of the protocol. In the initial phase, superpeers are selected randomly, as no reputation scores are available prior to the start of the FL phase (see Step 3). Each peer chooses another peer as a superpeer, and votes are then shared through flooding (Step 10), ensuring that all peers are aware of the most voted candidates, who will subsequently be selected as the roots.

A peer p can be rejected by a superpeer if the maximum number of children (MAX_CHILDS) is reached or if the superpeer detects a mismatch in the network round ($netr$) (Step 13), indicating desynchronisation, which is not permitted within a tree structure. Rejected peers must attempt to connect to other superpeers, excluding those already requested (Step 15), until one eventually accepts the connection. If no superpeer remains, the peer does not participate in the ML Phase and waits for the next Network Round to join again.

Algorithm 2 Superpeer Election Protocol

Require: $\mathbb{P}, \mathbb{NS}, MAX\_CHILDS, netr$

1: **for** $p \in \mathbb{P}$ **do**:
2: **if** $p_{netr} == 1$ **then**
3: Peer p randomly votes for one $n \in \mathbb{NS}$ to become superpeer;
4: **else**
5: $n \leftarrow \arg\max_{n \in \mathbb{NS}} \ reputation(n)$;
6: **if** $rep_n < rep_p$ **then**
7: Peer p vote on itself
8: **else**
9: Peer p votes on n ;
10: Peer p floods its vote to all other peers $p \in \mathbb{P}$;
11: Each peer p counts votes and selects the p most-voted peer;
12: Peer p try to connect to most-voted peer;
13: **while** $lenght([sp_{childs}]) \geq MAX\_CHILDS \vee netr_p \neq netr_{sp}$ **do**
14: Peer p does not connect to most-voted peer;
15: Peer p removes the most-voted from its local list of votes;
16: Each peer p counts votes and selects the most-voted peer;
17: Peer p try to connect to most-voted peer;
end

Once the reputation mechanism is introduced, superpeer selection is no longer random. Peers vote based on the reputation levels of other peers (Step 5), which are known across the network as reputations are calculated and propagated whenever updated. Each peer selects the neighbour with the highest reputation to be elected as a superpeer, choosing randomly in the event of a tie. Alternatively, a peer may nominate itself if it considers all neighbours less trustworthy than itself, i.e., if the highest-reputation neighbour has a reputation lower than its own (Step 6), introducing

proactivity and autonomy into the selection process.

4.3.3 Floding protocol

Since the P2P phase, detailed in Subsection 4.2.2, follows an unstructured topology, peers must rely on a flooding protocol to communicate with nodes beyond their immediate neighbours. The flooding protocol works by having a message, upon arrival at a peer, forwarded to all of that peer's neighbours to propagate it throughout the network. Each peer maintains a log of message IDs (which are randomly generated when a message is created) to avoid redundant transmissions, discarding any message it has already processed instead of forwarding it. The protocol concludes once all peers have received the message, that is, when the forwarding naturally ceases. This protocol enables full network communication through collaboration, eliminating the need for dedicated communication channels between every pair of peers and significantly reducing the complexity of network communications.

4.4 Final Remarks

As outlined in Section 4.1, FL-based intrusion detection systems continue to face significant challenges, notably data heterogeneity, susceptibility to adversarial behaviour, communication inefficiencies, and scalability limitations. The framework proposed in this dissertation was conceived to mitigate these issues and to narrow the gaps identified in the existing literature.

A central concern lies in managing non-IID data distributions, which are inherent to decentralised environments. This work introduces a personalised FL scheme in which each peer maintains a locally adapted model while contributing to the global aggregation. This design enables the coexistence of global consistency with local relevance, thereby alleviating the accuracy trade-offs commonly reported in prior studies.

System robustness is further enhanced through a decentralised reputation mechanism that penalises or excludes untrustworthy peers while promoting those that consistently behave reliably. By dynamically adjusting influence in the aggregation process, this mechanism strengthens resilience against poisoning attacks and faulty behaviour, safeguarding both the integrity and trustworthiness of the global model.

Finally, the proposed design addresses performance and scalability through a hybrid communication model. This approach reduces communication overhead and accommodates larger peer participation without compromising efficiency, resulting in a more scalable and adaptable architecture suitable for real-world deployment.

Chapter 5

Implementation

This chapter details the implementation steps undertaken to construct the proposed proof-of-concept system. Section 5.1 describes the system's architecture, outlining the structure and interactions of the core components within each peer. The discussion begins with the initialisation modules, which configure the network and ensure that all required preconditions are satisfied before execution. It then examines the implementation of both client-side and server-side components of each peer. In addition, auxiliary scripts and procedures that support or extend system execution are presented, emphasising their role in ensuring operational consistency and assisting the main components in fulfilling their functions. Section 5.2 introduces the tools and libraries employed during development, which facilitated the overall implementation process.

5.1 System's Implementation

This section explains the implementation of the system in detail, beginning with the initial preparation procedures required to set up the network. The focus then shifts to the core element of the architecture, the peer, describing its main components and internal logic. Finally, auxiliary utilities and support procedures that are essential for enabling the system's functionalities are described.

5.1.1 System's Preparation

Prior to the start of the system, two main aspects must be prepared: dataset splitting and the network structure. The dataset used, which will be detailed later, must undergo all the necessary steps, including combination, data cleaning, normalisation, correct labeling, and finally splitting.

Before execution, the dataset underwent a preparation phase, as detailed in Section 6.2.2, to ensure its quality and suitability for machine learning tasks. This phase, conducted by the dataset authors, included cleaning procedures to remove flawed or inconsistent entries, followed by feature normalisation to enhance data quality and facilitate model convergence. Each entry in the dataset is accurately labelled, representing either normal network traffic or a specific type of attack, with labels encoded numerically starting from 0 for normal traffic and incrementing according to the number of distinct attack types.

Algorithm 3 Dataset Preparation and Partitioning for Peer-based FL System

Require: $\mathbb{P}$

1: **Input:** List of network captures CSV files and corresponding label files
2: **Output:** Training and validation tensors for each peer
3: Initialize empty list `combined_data`
4: **for all** (data_file, label_file) in dataset **do**
5: Append cleaned and labelled data to `combined_data`
6: Split `combined_data` into `N` partitions (1 per peer)
7: **for** each peer p_i, where $i = 1 \dots |\mathbb{P}|$ **do**
8: Randomly assign a subset of data to p_i
9: Create `train_tensor` (features only) $\rightarrow$ store in `train/name(`p_i`).pt`
10: Create `val_tensor` (features + labels) $\rightarrow$ store in `validation/name(`p_i`).pt`

The data is distributed across multiple CSV files, as ilustrated in Algorithm 3, each corresponding to a different type of attack and accompanied by a matching label file. To simulate a non-IID distribution, which is essential for evaluating the robustness and personalisation capabilities of the proposed system, these datasets are combined in a non-uniform and randomised manner (Step 5). The resulting dataset contains all attack types, with samples of each attack distributed randomly. Consequently, any sequential subset of samples (e.g., samples 1 to 100) will contain an uneven mix of attack types and normal traffic, reflecting the heterogeneity typical of real-world network environments.

Once the combined dataset is constructed, it is partitioned into subsets based on the number of peers defined for the network (step 6). Each peer receives data of approximately equal size, but with significantly different distributions of attack types, attack sample quantities, and normal traffic, promoting local heterogeneity and the need for model personalisation. For each peer, two PyTorch tensor files are generated: one stored in the train folder, which contains only the features and is used for local training; and one stored in the validation folder, which includes both features and labels and is used for evaluation and metric computation (steps 9 and 10, respectively). These files follow a consistent naming convention aligned with the container names, ensuring clarity during deployment. The destination folders for these files are stored in Docker volumes, which serve as persistent data stores for containers. When a volume is mounted, the corresponding directory becomes accessible within the container; unlike bind mounts, volumes are managed by Docker and isolated from the host system's core functionality, providing a controlled and reliable storage solution. During execution, the data is loaded into memory through PyTorch DataLoader instances, enabling efficient training and evaluation of local models.

Regarding the network structure, and considering the proof-of-concept nature of this work, the topology is generated with predefined peer connections, meaning that each peer is aware of its neighbours prior to execution. While this does not reflect the most realistic deployment scenario, it serves the purpose of enforcing decentralisation and preventing excessive connectivity between nodes. A maximum number of neighbours is configured to maintain a controlled and sparse topology. Although the connections are statically defined, the network graph is randomly generated at

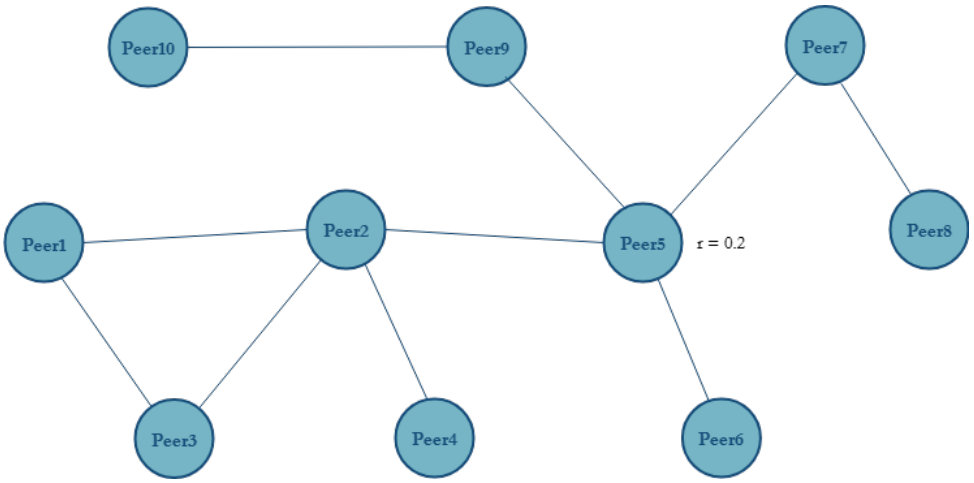

Figure 5.1: Example of the P2P network organisation represented as a graph

each setup (see example in Figure 5.1).

The `gen_network.py` script is responsible for this process. It constructs a JSON file (see Figure 5.2) representing the network topology, where each entry corresponds to a peer, including its unique identifier and the list of neighbours. As connections are bidirectional, each link is recorded from both endpoints to accurately reflect the undirected structure.

```
{
   "peers": {
        "peer1:50000": {
            "id"         : 7350483298983324612,
            "neighbours" : [
                           "peer5:50004",
                           "peer2:50001",
                           "peer8:50007"
            ]
        },
        "peer2:50001": {
            "id"         : 3221317979898994507,
            "neighbours" : [
                           "peer7:50006",
                           "peer5:50004",
                           "peer1:50000"
            ]
        }
    }
}
```

Figure 5.2: Example of a peer network configuration in JSON format

At runtime, each peer reads this file during initialisation to identify and establish communication with its assigned neighbours. The network generation process can be repeated for different test scenarios, allowing variations in structure by modifying parameters such as the total number of

peers or the minimum and maximum number of neighbours per node. This flexibility enables the creation of diverse network configurations to evaluate the behaviour of the system under different levels of randomness and connectivity.

5.1.2 Peer Client Architecture

On the client side of each peer is the set of functionalities responsible for all external communications, as shown in Figure 4.2. This module handles every outbound interaction, including requests to other peers and coordination within the network. It is also where key processes such as the election of the superpeer, local model training, evaluation of the aggregated model, and the calculation of the superpeer's reputation are conducted.

Two central procedures govern the transitions between the system's phases, as illustrated in Algorithm 4: *P2PToStruct()* and *StructToP2P()*. The first, as the name suggests, governs the transition from the P2P topology to a structured, tree-like hierarchy. Initially, each peer assesses its neighbours and votes for the one it deems most trustworthy to act as a superpeer, a process discussed in detail in Section 4.2.2. After casting its vote, the peer broadcasts it through the network using the flooding protocol described in Section 4.3.3. The peer then waits in the *GatherVotes()* (step 4) function until it has received a sufficient number of votes from the network. Once this threshold is reached, the peer identifies the candidate with the highest number of votes and attempts to connect as a superpeer, selecting randomly if multiple peers share the highest count. If the connection is successful and the superpeer is available, the peer notifies the rest of the network of its status, enabling others to proceed accordingly (step 8). This iterative process continues until the entire network has transitioned into the structured phase.

Once this transition is complete, the *StructToP2P* procedure takes over, managing the FL operations. In this phase, the peer participates in the training and evaluation cycle of the distributed learning system. Depending on the current round, it may initialise training either with the model shared by the superpeer or with its own local version, as detailed in Section 4.2.3. Training is performed locally over a predefined number of rounds and epochs. Upon completion, the peer transmits its contribution to the superpeer, which aggregates the received models and sends back a global version.

The peer then evaluates (step 18) this aggregated model using its local validation set. Based on the model's accuracy and the superpeer's historical performance, a new reputation score is computed. If the model demonstrates acceptable performance, its parameters are adopted locally. However, if the model's quality is insufficient or raises suspicion of malicious behaviour, the peer may trigger a termination of the federated learning process, refusing further collaboration (step 21).

5.1.3 Peer Server Architecture

On the server side, which resides within each peer process, the functionalities are primarily structured around gRPC-based endpoints. These endpoints are responsible for handling inbound re-

Algorithm 4 Client-Side Peer Logic

```
1: function P2PToStruct
2:     selected_superpeer ← VoteSuperpeer
3:     Broadcast selected superpeer vote
4:     votes ← GatherVotes(timeout)
5:     if votes.count ≥ votes threshold then
6:         elected ← select_winner(votes)
7:         if ConnectToSuperpeer(elected) == success then
8:             Broadcast settled status
9:     Wait until all peers are settled
10: function StructToP2P
11:     if current_round == 0 then
12:         model ← Get superpeer model
13:     else
14:         model ← Initialise local model
15:     TrainModel(model, epochs)
16:     SendModel(model)
17:     aggregated ← Received aggregated model
18:     accuracy ← Evaluate(global model)
19:     UpdateReputation(accuracy)
20:     if accuracy < acceptance_threshold then
21:         EndMlPhase()
22:     else
23:         Adopt global model
```

quests, processing client-side data, and coordinating communication between peers. Since peers can act either as regular participants or as superpeers, the server-side logic is designed to support both roles and facilitate transitions between them.

The procedures implemented on the server are diverse, supporting a wide range of coordination, control, and reputation-related operations. Key endpoints include:

- *GetVotes()*: Returns the set of votes currently known by the peer, enabling others to retrieve distributed election data.
- *SetRole()*: Assigns the operational role of the peer, either remaining as a standard peer or being elevated to superpeer status.
- *PeerInfo()*: Provides status metadata about the peer, including its current role and network state.
- *ResetPeerStruct()*: Resets internal data structures, including child parameters, child connections, and reputation lists, preparing the peer for a new phase or state. This method is invoked whenever the tree is dissolved or the new phase requires empty data structures to operate.
- *GetTrainingStatus()*: Allows querying the peer to determine whether the training process has concluded or is still ongoing.

- *SubmitVote()*: Submits a vote for superpeer selection and disseminates it across the network using a flooding-based protocol. To avoid message flooding, a control mechanism tracks message propagation to ensure each message is processed only once.
- *BroadCastRep()*: Broadcasts reputation information known to the peer, supporting synchronisation of trust data across the network.
- *EnterMLPhase()*: Initiates the machine learning phase. During this phase, the peer (if acting as a superpeer) collects model updates from clients, performs aggregation, and calculates reputations. Reputation scores, as detailed in Section 4.3.1, influence the weight of each contribution. Contributions from peers with insufficient reputation may be discarded for the remainder of the phase.
- *EnterTree()*: Handles the transition of a peer into the superpeer's structured tree topology. Peers may be rejected due to unsynchronised state (response code 202) or capacity constraints (response code 403). Acceptance is confirmed with a successful response.

Together, these endpoints orchestrate the peer's behaviour during both the topology formation and FL phases. They also provide essential support for model coordination, contribution filtering, and trust-based interaction, embedding core functionalities of the system's decentralised and reputation-aware architecture. All message formats and endpoint definitions are formally specified in the associated `.proto` file used to generate the gRPC service interfaces.

5.1.4 Utilities

For the system to function correctly, several supporting procedures and utilities are employed across both the client and server sides. These functionalities are encapsulated in a set of auxiliary scripts, each responsible for specific runtime behaviours such as automation, communication handling, logging, and training support. Among them, two scripts are particularly central to the overall system: `aux_func.py` and `ml_aux_func.py`.

The `aux_func.py` script gathers a broad range of general-purpose procedures used throughout the system. These include functions such as *gen_id()*, which generates unique identifiers for messages and votes, and *get_most_voted()*, which retrieves the most voted peer, either randomly or based on the vote distribution. The script also provides utilities to construct and dispatch gRPC messages, load datasets into PyTorch-compatible DataLoaders for training, and log runtime information to external files, which are essential for debugging, auditing, and post-execution analysis. These functions are used by both server and client components of the peer nodes, making the script a shared foundation for multiple system operations.

In parallel, `ml_aux_func.py` focuses specifically on ML-related processes. It implements all local training procedures, including support for epochs, rounds, and configurable optimisers. The script also provides functions to manage model weights and reputation updates, adjusting them based on training performance metrics such as accuracy. Furthermore, it supports model

personalisation by preserving the model head while fine-tuning only the feature extractor layers, thus maintaining peer-specific characteristics. A key function in this script is the model aggregation procedure, which combines client models using a Federated Averaging (FedAvg) approach to produce a global model that reflects distributed knowledge. Additionally, the script contains tools for model evaluation and result logging, which are essential for tracking performance over time and conducting later-stage analysis, as discussed in the following chapter.

Other supporting files also play important roles in the system:

- `models.py` provides a collection of pre-defined ML architectures that can be extended or replaced with more sophisticated and robust models as needed.
- `net_vis.py` enables real-time visualisation of the network topology, allowing users to inspect peer connections and monitor dynamic changes in the structure through graph-based representations.
- `constants.py` defines all global macros and configuration parameters for the system. This includes the size of the network, the number of connections per peer, the type of tests to be performed (e.g., with or without personalisation or reputation mechanisms), and the setup of adversarial peers. By adjusting this file, users can flexibly control the system's behaviour without modifying core code.

5.2 Tools and Libraries

This section describes the principal tools and libraries that enabled the implementation of the proposed system. These resources significantly contributed to a faster and more robust development process, offering essential functionalities that span across various aspects of the system. Specifically, they supported activities such as network deployment, communication protocol management, ML model construction, and dataset preparation.

5.2.1 Docker

Docker is a platform-as-a-service (PaaS) solution that simplifies application deployment using lightweight containers [13]. These containers rely on OS-level virtualisation, allowing multiple isolated instances to share the same operating system kernel. Each container bundles the application with its required dependencies, ensuring portability and consistent execution across environments.

Docker supports deployment on Linux, Windows, and macOS systems. While it runs natively on Linux, it relies on a virtual machine layer on other platforms to ensure compatibility. By leveraging Linux features such as namespaces and control groups (cgroups), Docker isolates containers with minimal resource overhead, enabling faster start-up times, higher application density per host, and greater efficiency compared to traditional virtual machines [10].

In this solution, Docker is employed to package and deploy each peer of the system, with the network deployed on single machines to simulate system behaviour by launching all peers locally. This approach ensures reproducibility, simplifies dependency management, and facilitates distributed testing, while the isolation of peer components within containers further supports modular development and enables reliable execution with reduced deployment complexity.

5.2.2 GRPC

gRPC [26] is a high-performance, open-source remote procedure call (RPC) framework initially developed by Google. It enables efficient communication between distributed components using HTTP/2 as the transport protocol and Protocol Buffers as the interface definition language. gRPC supports advanced features such as authentication, bidirectional streaming, flow control, and deadline propagation, and offers code generation for multiple programming languages. Due to its use of HTTP/2, gRPC achieves low latency and high throughput, making it suitable for microservice architectures and client-server communication across diverse environments.

In this work, gRPC is employed to manage communication between peers in a lightweight and efficient manner. A proto file (`.proto`), shown as an example in Figure 5.3, is first created to define the data structure schema (messages) and requests (services), establishing a precise specification for system communication. Once defined, the implementation of the communication modules is simplified, as the Python gRPC library (grpcio), together with the protocol buffer compiler protoc, automatically generates the required code, enabling the creation of channels, stubs, and the management of requests. This integration ensures minimal network overhead, allowing system performance to be primarily driven by the ML tasks rather than inter-peer communication.

5.2.3 Programming language & Libraries

The implementation of this solution was developed in Python, using version 3.11. Python was chosen due to its readability, wide adoption in ML and distributed systems, and extensive ecosystem of libraries. Throughout the development, several libraries were employed to simplify complex tasks and promote code reuse.

- **scikit-learn (sklearn) [50]** – A widely used, open-source Python library for ML, offering an extensive suite of efficient tools for predictive data analysis, including classification, regression, clustering, dimensionality reduction, model selection, and evaluation. It features a clean, consistent API built on top of NumPy and SciPy and supports transformations such as cross-validation, pipelines, and hyperparameter tuning.

- **logging [57]** – A standard Python library providing a flexible and configurable framework for emitting log messages during program execution. It supports various severity levels, output handlers, and message formatting, making it indispensable for debugging, monitoring, and auditing distributed systems.

```
syntax = "proto3";

service NetworkService {
  rpc Ping              (Message)       returns      (Message);
  rpc Echo              (Message)       returns      (Message);
  rpc GetVotes          (Empty)         returns      (VoteList);
  rpc SubmitVote        (Vote)          returns      (Vote);
  rpc SetRole           (Role)          returns      (Empty);
  rpc ResetPeerStruct   (Empty)         returns      (Message);
  ...
}
message Role {
  int64  message_id    = 1; // Unique message identifier
  int64  sender_id     = 2; // ID of the sender
  int64  role_id       = 3; // 1 for sp, 2 for peer
  int32  network_round = 4;
  int64  timestamp     = 5; // Timestamp
}
...
```

Figure 5.3: Excerpt of Solution's Proto File

- **TensorFlow [2]** – An end-to-end open-source platform for ML developed by Google, which facilitates the construction, training, and deployment of models across a wide range of environments, including desktops, servers, mobile devices, and embedded systems. In this solution, TensorFlow is employed to manage datasets, create models, and perform machine learning operations, including training and evaluation.

- **threading [8]** – A module in Python's standard library enabling concurrent execution using threads. This is particularly useful for managing parallel tasks such as network I/O and background computation without blocking the main execution flow.

- **random [23]** – Provides routines for pseudorandom number generation, sampling, and shuffling. Commonly employed for experiments that require controlled randomisation, such as non-IID dataset partitioning or stochastic behaviour modelling.

- **time [9]** – A lightweight standard library module offering time-related operations like measuring elapsed time, adding delays, and timestamp generation. Essential for performance profiling, debugging timeouts, and synchronising distributed operations.

- **json [31]** – A core module for parsing and serialising JSON (JavaScript Object Notation) data. Essential for configuration management, inter-process communication, and data exchange in a format that is both human-readable and language-agnostic.

- **NumPy [28]** – The foundational package for numerical computation in Python. It provides the powerful `ndarray` object for handling multi-dimensional arrays, alongside vectorised

operations, broadcasting, random sampling, and linear algebra routines.

- **pandas [48]** – A high-level data manipulation and analysis library built upon NumPy. It introduces DataFrame and Series structures, which are crucial for preprocessing tasks, including data transformation, grouping, cleaning, and integration with ML pipelines.

Chapter 6

Experimental Evaluation

In this chapter, the experimental methodology adopted to evaluate the proposed solution is presented. The focus is on assessing key properties of the system, namely decentralisation, personalisation, and robustness. To this end, a large and heterogeneous dataset was employed, with particular emphasis on the non-IID data distribution, which constitutes a central concept addressed in this dissertation.

The chapter is structured as follows. First, the research questions guiding the evaluation are introduced. This is followed by a detailed description of the experimental setup, including the metrics selected for performance assessment, the dataset characteristics, and the computational environment used. Finally, the conducted experiments are described, outlining the configurations applied, such as the number of rounds, architectural variations, and the functionalities under examination, together with the results obtained for each test case.

6.1 Evaluation Questions

In this section, the research questions guiding the experimental process are outlined. These questions are designed to evaluate the proposed system with respect to its ability to address the main challenges identified in the literature, namely non-IID data distribution, personalisation, and robustness against adversarial behaviour. The questions are as follows:

- **Q1:** How does the proposed system perform under the non-IID data scenario? Specifically, does the global model maintain satisfactory performance when exposed to heterogeneous and diverse attack types?

- **Q2:** To what extent does the personalisation feature contribute to improving the performance of local models?

- **Q3:** How resilient is the proposed system when facing faulty or malicious participants?

6.2 Experimental Environment

In this section, the elements that compose the experimental setup are presented in detail. First, the metrics considered for evaluation are described, ensuring a comprehensive and rigorous basis for the analysis. These metrics serve as the foundation for assessing the system's performance in terms of accuracy, robustness, and efficiency. Subsequently, the dataset employed in the experiments is introduced, with emphasis on its characteristics, diversity, and the justification for its selection, as it plays a fundamental role in validating the results. Finally, the computational environment is detailed, specifying the hardware and software configurations in which the experiments were conducted. This description is crucial not only to support the validity of the results but also to provide a clear reference for future replications of the experimental process.

6.2.1 Metrics

Several metrics are used to evaluate the system's performance. One of these is the Confusion Matrix, a simple table that measures how well a classification model performs by comparing its predictions with the actual results, highlighting where the model was correct or incorrect.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

Table 6.1: Confusion Matrix Diagram

The elements of the confusion matrix (Table 6.1) are interpreted as follows:

- **True Positives (TP):** Instances correctly identified as attacks.
- **True Negatives (TN):** Instances correctly identified as non-attacks.
- **False Positives (FP):** Instances incorrectly classified as attacks.
- **False Negatives (FN):** Instances incorrectly classified as non-attacks.

Based on the confusion matrix, the following evaluation metrics can be calculated:

- **Precision (P):** The fraction of correctly identified attacks among all instances flagged as attacks.

$$P = \frac{TP}{TP + FP} \tag{6.1}$$

- **Accuracy (ACC):** Represents the overall correctness of the system by measuring the proportion of correctly classified instances.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.2}$$

- **True Positive Rate (TPR) or Recall:** The proportion of actual attacks that are correctly classified.

$$TPR = \frac{TP}{TP + FN} \tag{6.3}$$

- **F1-Score (F1):** The harmonic mean of Precision and Recall, providing a single measure that balances both metrics.

$$F1 = \frac{2 \cdot P \cdot TPR}{P + TPR} \tag{6.4}$$

6.2.2 Dataset Description

The dataset used in this work is the Kitsune Network Attack Dataset [44], a comprehensive collection of nine network attack datasets captured from either an IP-based commercial surveillance system or a network composed of IoT devices. Each dataset comprises millions of network packets and encompasses a variety of cyber attacks typically observed in real-world network environments.

Attack Type	Attack Name	Tool	Description: *The attacker...*	Violation	Vector	# Packets	Time [min.]
Recon.	**OS Scan**	Nmap	*...scans the network for hosts, and their operating systems, to reveal possible vulnerabilities.*	**C**	1	**1,697,851**	52.2
	Fuzzing	SFuzz	*...searches for vulnerabilities in the camera's web servers by sending random commands to their cgis.*	**C**	3	2,244,139	85.5
Man in the Middle	**Video Injection**	Video Jack	*...injects a recorded video clip into a live video stream.*	**C, I**	1	2,472,401	33.4
	ARP MitM	Ettercap	*...intercepts all LAN traffic via an ARP poisoning attack.*	**C**	1	2,504,267	28.2
	Active Wiretap	Raspberry PI 3B	*...intercepts all LAN traffic via active wiretap (network bridge) covertly installed on an exposed cable.*	**C**	2	4,554,925	95.6
Denial of Service	**SSDP Flood**	Saddam	*...overloads the DVR by causing cameras to spam the server with UPnP advertisements.*	**A**	1	4,077,266	40.8
	SYN DoS	Hping3	*...disables a camera's video stream by overloading its web server.*	**A**	1	2,771,276	52.8
	SSL Renegotiation	THC	*...disables a camera's video stream by sending many SSL renegotiation packets to the camera.*	**A**	1	6,084,492	65.6
Botnet Malware	**Mirai**	Telnet	*...infects IoT with the Mirai malware by exploiting default credentials, and then scans for new vulnerable victims network.*	**C, I**	**X**	764,137	118.9

Figure 6.1: Kitsune Network Attacks Dataset [44]

Each attack dataset is provided in three formats: a preprocessed CSV file suitable for machine learning, a corresponding label CSV file indicating benign or malicious packets, and the original network capture in pcap format. The preprocessed CSV files contain m rows, representing individual network packets captured chronologically, and 115 features extracted per packet. These features are generated using the AfterImage feature extractor, which produces a rich statistical snapshot of network activity. For each packet, the feature vector captures both host-level and channel-level traffic statistics, including traffic originating from the packet's source MAC and IP address (SrcMAC-IP), the source IP (SrcIP), the communication channel between the source and destination IPs (Channel), and the source-destination TCP/UDP socket (Socket). To capture temporal dynamics, the feature extractor computes the same 23 base statistics over five exponentially weighted time windows, ranging from 0.01 seconds to 5 seconds, resulting in a total of 115 features per packet. In cases where a feature is not applicable, such as a missing socket for non-TCP/UDP traffic, the corresponding feature values are zeroed. Consequently, each packet is consistently represented as a vector in $\mathbb{R}^{115}$, suitable for use in ML models.

As illustrated in Figure 6.1, the dataset includes a diverse range of network attacks, categorised into reconnaissance, man-in-the-middle, denial-of-service, and botnet malware attacks. Examples include OS scans and fuzzing for reconnaissance, ARP MitM and video injection for MitM attacks, SSDP and SYN floods for denial-of-service attacks, and Mirai for botnet malware. Each dataset begins with a baseline of benign network traffic, followed by the initiation of the respective attack. Detailed labels indicate the status of each packet, allowing supervised learning and model evaluation.

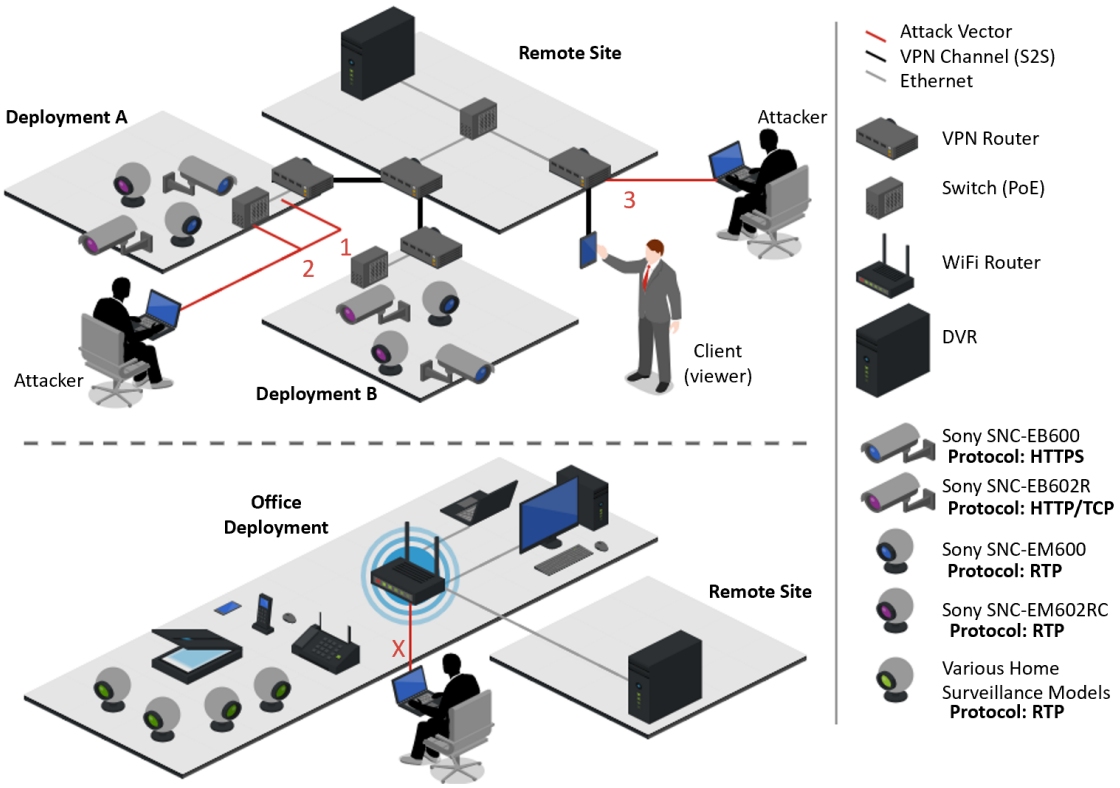

Figure 6.2: Kitsune Data Collection Schema [44]

Data collection, as presented in Figure 6.2, was performed at specific points within the network, capturing traffic both prior to and during attacks. The preprocessed CSV files provide a ML-ready representation, while the original pcap captures allow for potential feature engineering or verification of preprocessing steps. This multivariate, sequential, time-series dataset enables a comprehensive evaluation of models in realistic and heterogeneous network conditions, with 27,170,754 packet instances in total.

The structured organisation of the dataset, with separate directories for each attack containing the feature matrix, labels, and original pcap capture, facilitates efficient partitioning and processing. This organisation supports the experiments in this dissertation, particularly those addressing the challenges posed by non-IID data.

6.2.3 Computational Environment

To execute the proposed system with the computational resources required, the experiments were conducted on the *Cluster S* provided by FCUL, composed of 16 nodes. Each node corresponds to a Dell PowerEdge R410 server, equipped with two Intel® Xeon® E5520 processors running at 2.27 GHz. Each processor provides four physical cores with Hyper-Threading support, resulting in a total of eight threads per CPU. Considering the entire cluster, this configuration amounts to 32 CPUs, 128 physical cores, and 256 threads available for parallel execution. Every node is equipped with 32 GB of DDR3 memory running at 1066 MHz, supporting the computational requirements of distributed training.

In terms of storage, each node is provisioned with a 146 GB SAS disk operating at 15,000 RPM. Additionally, some nodes provide extended storage capabilities. Nodes s1 to s4, the latter of which is used in this dissertation, are equipped with two additional 3 TB SATA disks, while nodes s9 to s12 include an extra 120 GB SSD. Network connectivity is ensured by dual Gigabit Ethernet interfaces, provided through both Broadcom NetXtreme II BCM5716 and Intel Gigabit controllers, supporting full-duplex communication and hardware-level optimisations. The cluster is interconnected through a Gigabit Ethernet backbone, ensuring efficient parallelism and distributed resource allocation across the infrastructure.

6.3 Experimental Tests

6.3.1 Configuration

The system includes a configuration file (`constants.py`) that specifies all initial conditions and features for execution. Adjusting these configurations can significantly impact network behaviour. Table 6.2 lists all configurable variables, their descriptions, and the values used in the experimental tests.

6.3.2 Non-IID Data Challenge

As introduced in Question 1, the objective is to evaluate the effectiveness of the proposed solution under non-IID data distributions. As detailed in Section 5.1.1, the subdatasets are randomly mixed so that attack types and quantities vary across peers, approximating real-world conditions; the combined dataset is then partitioned and distributed among all peers.

To assess multi-class classification under non-IID conditions, three experimental configurations were prepared: (i) a binary setting with the Mirai Botnet attack and normal traffic, (ii) a three-class setting comprising Mirai Botnet, OS Scan and ARP MitM attacks plus normal traffic, and (iii) a multi-class setting including all attack types present in the dataset. These experiments evaluate both the system's capacity to discriminate different attack types and its overall ability to mitigate the challenges introduced by non-IID data.

Figure 6.3 summarises the results for each configuration. It includes scatter plots showing both the mean accuracy and the individual client model accuracies, reflecting the influence of

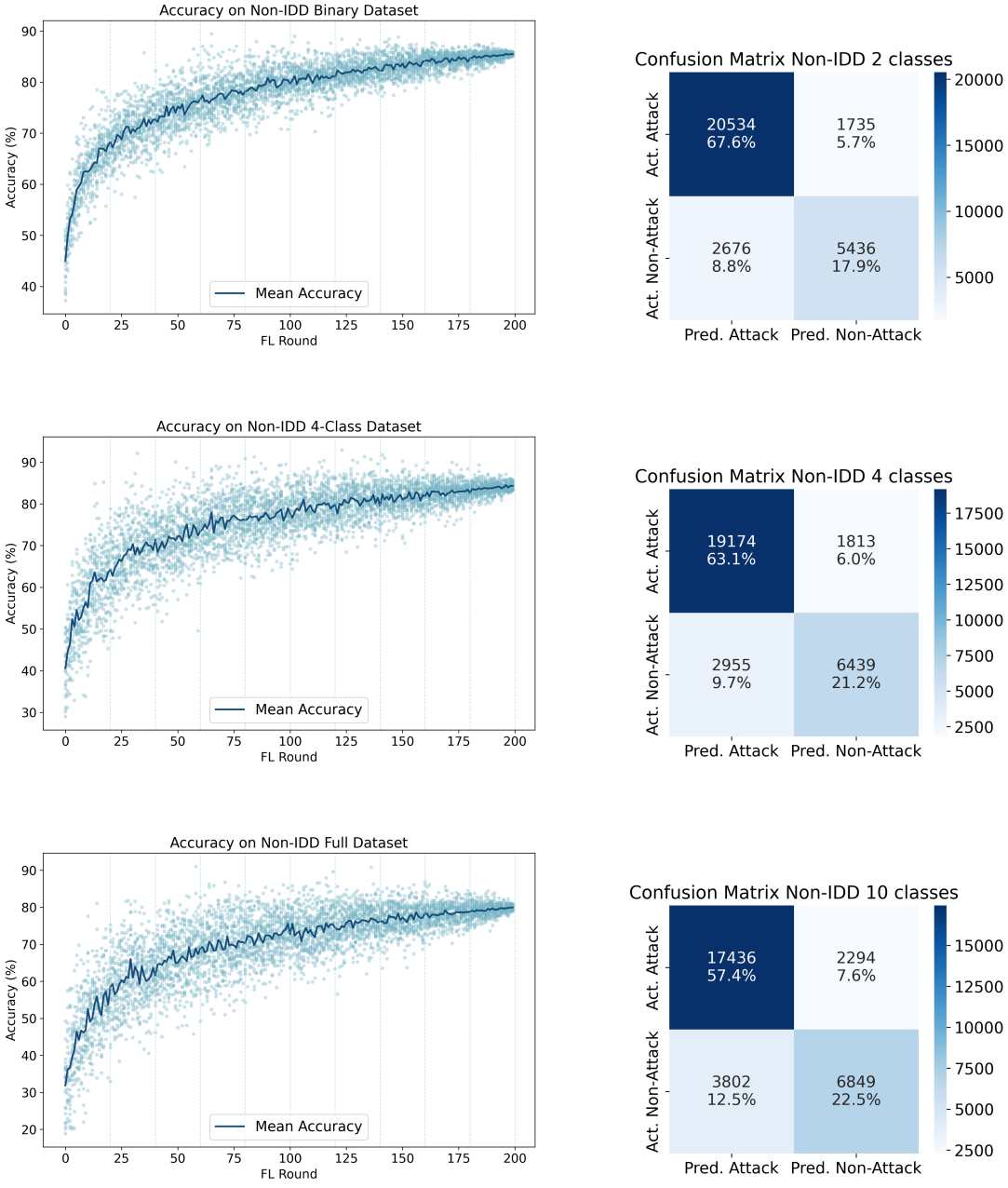

Figure 6.3: Accuracy and confusion matrix plots of the model evaluated on dataset variations with 2, 4, and all classes

Variable	**Description**	**Value**
$\|\mathbb{P}\|$	Number of Peers	25
$\|\$\mathbb{P}\|$	Number of Superpeers	5
MAX_CHILDS	Maximum Number of Peers per Superpeer	4
POLL_INTERVAL	Seconds between polling network state	10
NETWORK_ROUNDS	Number of network rounds	10
INITIAL_REP_LEVEL	Initial reputation value	0.5
EPOCHS	Number of Epochs	5
ROUNDS	Number of FL Rounds	20
BATCH_SIZE	Batch size	64
REP_THRESHOLD	Minimum Reputation Threshold	0.6
LR	Learning Rate	0.001
SMOOTH_FACTOR	Smooth Factor (Reputation)	0.9

Table 6.2: System Configuration Descriptions and Values

the global model at a given FL round. The Figure also presents confusion matrices, showing the average number of samples correctly and incorrectly classified as positive or negative. Starting with the binary-class dataset, the mean accuracy begins around 45% and rapidly increases during the first 25 FL rounds. The evolution follows a logarithmic trend, with accuracy stabilising over the last 50 rounds. Dispersion across peers is higher in the early rounds, where the curve's slope is steeper, and gradually decreases as peers converge to a mean accuracy of approximately 85%.

The confusion matrix provides further insight into system performance under the binary-class setting. Out of 30,381 samples, around 20,500 were correctly classified as attacks, corresponding to roughly two-thirds of the total.

For the 4-class dataset, a similar trend is observed. The model starts at approximately 40% accuracy and reaches 84% by the end of the FL rounds. Dispersion is somewhat more irregular, particularly during the first 100 rounds, reflecting increased variability among peers. The confusion matrix indicates strong performance, with the system correctly predicting nearly 19,000 positive samples and around 6,500 negative samples.

The 10-class dataset presents a more challenging scenario. The model begins at roughly 30% accuracy but evolves to around 80% by the end of the FL rounds, representing the largest increase among the three experiments. Dispersion is highest in this case, reflecting a more uncertain and complex learning process due to the increased diversity of data. The confusion matrix demonstrates the system's ability to correctly classify attacks and non-attacks, although misclassifications are slightly more frequent compared to the previous datasets.

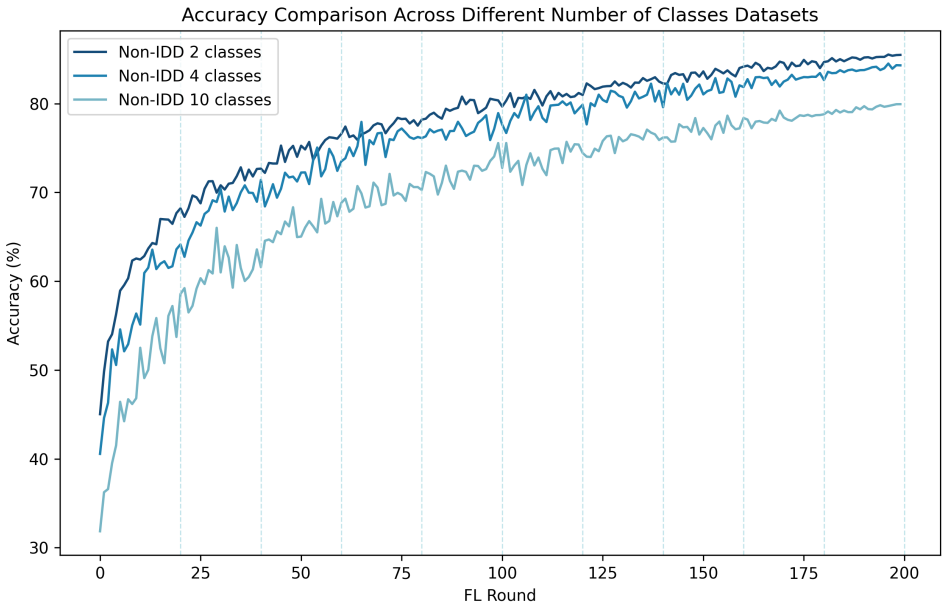

Figure 6.4: Accuracy comparison of the model on dataset variation with 2, 4, and all classes

Figure 6.4 illustrates how dataset complexity and Non-IID data affect system performance. The model performs best on the binary-class dataset, showing smoother evolution and less variability. The 4-class scenario follows closely, exhibiting slightly greater dispersion due to the additional classes. Despite the challenges posed by the 10-class dataset, the system achieves substantial improvement throughout the 200 FL rounds, reaching approximately 80% accuracy. This indicates that while the complexity of the data introduces variability, it does not impede the overall effectiveness of the model or the collaborative contribution of peers in achieving a strong global model.

6.3.3 Personalisation Feature

The second research question addressed in this evaluation concerns the impact of personalisation on system performance under a Non-IID, multi-class dataset. To assess this, the system was executed both with and without the personalisation feature on a 10-class dataset, and the accuracies of the local models were recorded for comparison.

Figure 6.5 presents the results obtained from the non-personalisation run. The evolution of the mean global model accuracy is noticeably noisier and more irregular, with higher dispersion present in almost every FL round. The mean accuracy starts at 30%, and the progression curve shows a shallower slope, indicating a more difficult and unstable learning process. By the end of the FL rounds, the model stabilises at approximately 70% accuracy. The confusion matrix in the Figure shows around 13,300 samples correctly classified as attacks and 8,031 samples correctly classified as non-attacks, while also evidencing a substantial gap between correct and incorrect predictions.

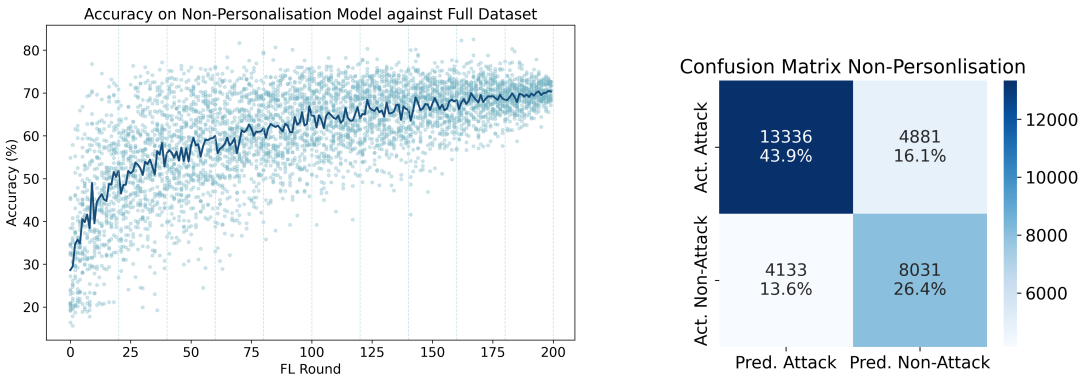

Figure 6.5: Accuracy and confusion matrix plots of the non-personalised model

Figure 6.6 compares the results with and without personalisation. The difference is evident both in the progression of accuracy and in the final performance. Without personalisation, the evolution is slower and more unstable, with frequent regressions to lower accuracy levels. For example, a mean accuracy of 60% is reached within the first 25 rounds under personalisation, while the non-personalised version requires nearly 80 rounds to achieve the same result. At the end of the system execution, the personalised approach reaches a mean accuracy of about 80%, whereas the non-personalised run remains almost 10 percentage points lower. These results clearly highlight the importance of personalisation in handling Non-IID data, as it enables the model to adapt to each client's local distribution rather than applying a single global model uniformly across heterogeneous scenarios.

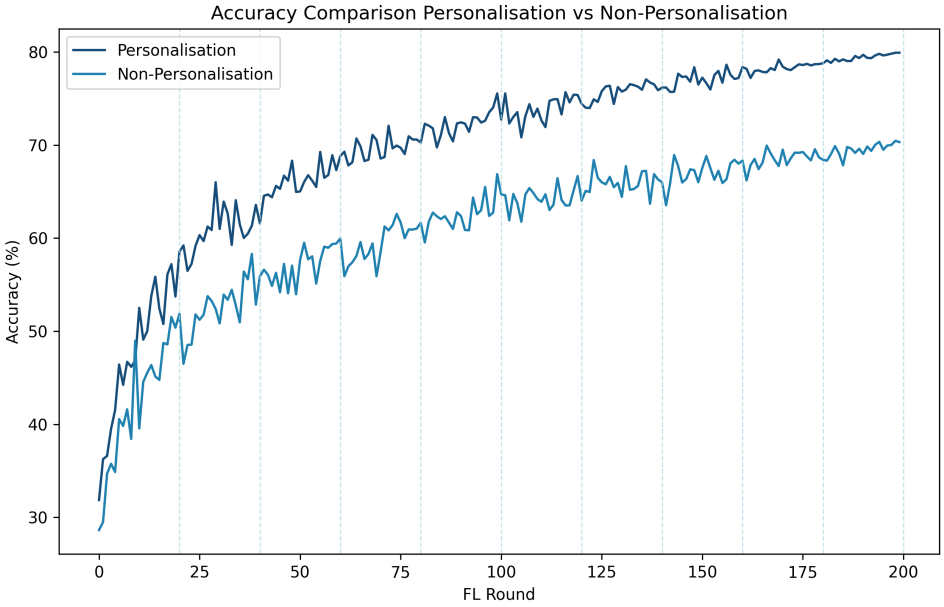

Figure 6.6: Accuracy comparison between personalised and non-personalised models

6.3.4 Reputation Feature

The third research question to be addressed in this evaluation concerns the robustness of the system against malicious or faulty participants. In this solution, as previously explained, a reputation mechanism was introduced to mitigate such challenges. Therefore, in a manner similar to Subsection 6.3.3, the system will be executed with and without the reputation feature enabled.

To simulate adversarial behaviour, the system was executed with approximately five faulty peers. This was achieved by preventing these participants from updating their local models, so they retained their initial, default models without any learning throughout all rounds of the system. Nevertheless, they operated as regular peers, thereby exerting a negative influence on the system across the rounds.

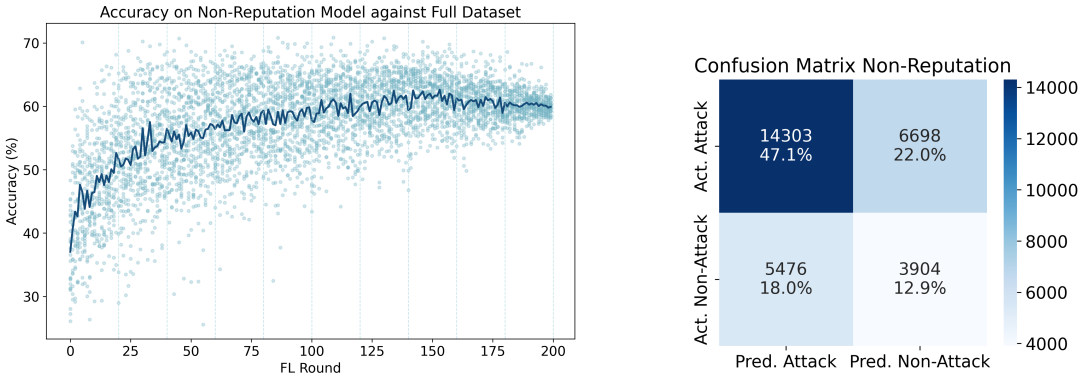

Figure 6.7: Accuracy and confusion matrix plots of the model evaluated on dataset variations with 2, 4, and all classes

Figure 6.7 illustrates the behaviour of the system without the reputation mechanism. The evolution of the mean accuracy is slow and irregular, with frequent fluctuations between consecutive FL rounds. The variations in accuracy across rounds are substantial, indicating instability in the global model's performance. Dispersion among peers is also pronounced, with accuracy gaps exceeding 50% in some cases. The trend shows initial improvement during the first 100 FL rounds, followed by a plateau and eventual degradation in the final 50 rounds, where the mean accuracy drops below 60%. These results demonstrate that the presence of faulty or malicious clients not only hinders the overall progress of the system, but can also actively degrade performance by contributing poor updates or withholding relevant features from their local datasets, which are not available in other peers.

Figure 6.8 compares the results of a solution employing a reputation mechanism with one that does not. Although both runs start from a similar mean accuracy, their evolution diverges significantly. The reputation-based model demonstrates a clear advantage, showing greater robustness to faulty peers. While the influence of malicious or faulty clients is still present, it is considerably constrained, allowing the global model to continue improving without entering a plateau or suffering degradation. This highlights the importance of the reputation mechanism, particularly in scenarios where the proportion of faulty or malicious peers is large or represents a significant part

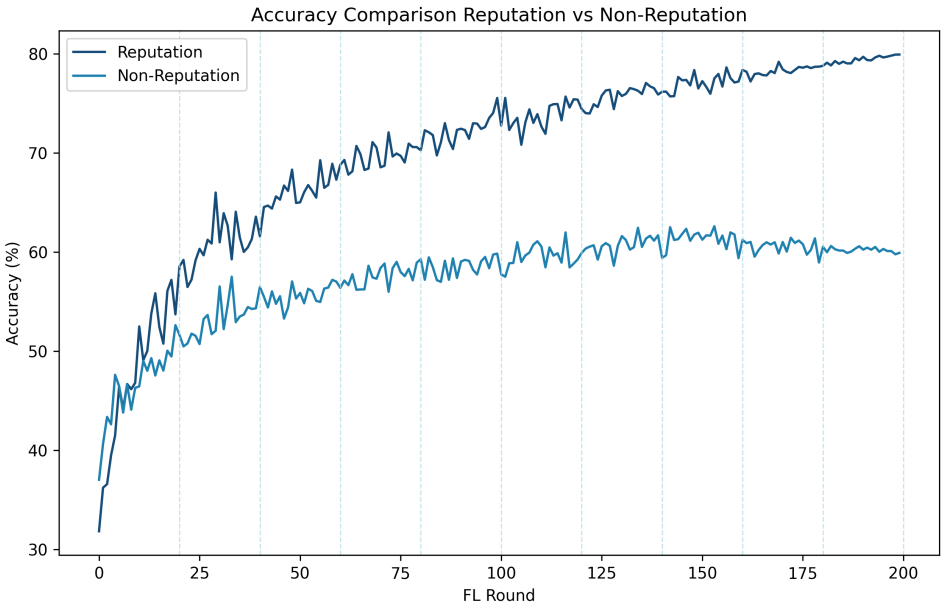

Figure 6.8: Accuracy comparison between reputation-based and non-reputation based models

of the network.

6.3.5 Final Remarks

In this section, the solution was evaluated under three scenarios: handling Non-IID data, with the personalisation feature, and with the reputation feature. The results show that the system experiences a slight degradation in accuracy when increasing the number of classes for classification, which is expected in such scenarios. Nevertheless, it maintains an overall accuracy of approximately 80%, with some clients achieving even higher results. This indicates that the system is capable of handling real-world scenarios, where multi-class classification is common.

Regarding the additional features, these prove to be fundamental to the robust and adaptive nature of the proposed solution. Without them, the system loses performance and reduces its ability to correctly detect and classify attacks in the evaluated samples. Figure 6.9 highlights that, although both features are important, the absence of the reputation mechanism has a stronger negative impact than the absence of personalisation. Without reputation, the system is highly affected by faulty and malicious peers, which is considerably more detrimental to performance than the lack of personalisation. This effect becomes even more evident when the proportion of malicious peers increases, as they can not only halt the learning progression of the global model but also actively degrade it through harmful contributions. This is reflected in the single execution in which its trend curve shows a decreasing pattern in the later FL rounds.

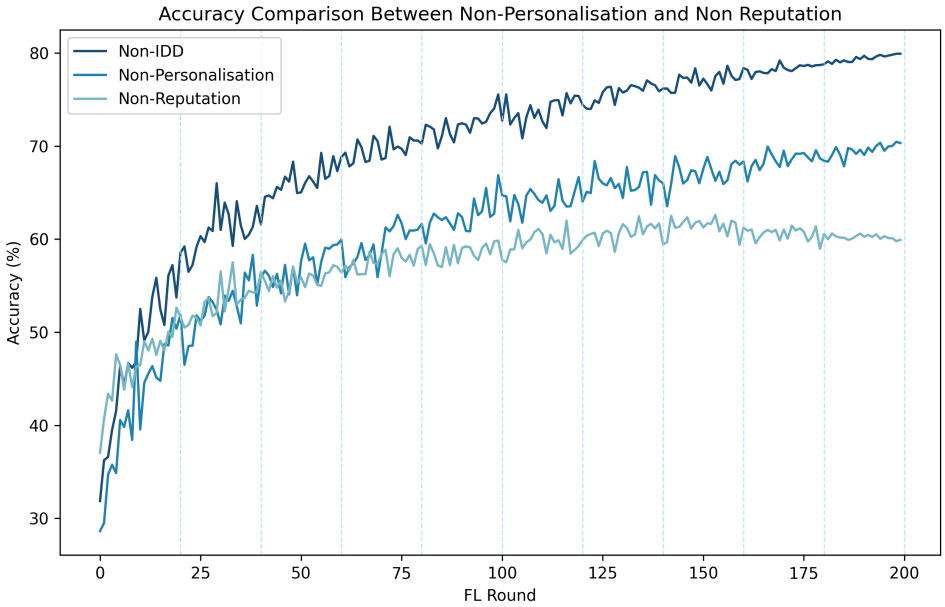

Figure 6.9: Accuracy comparison between non-personalised and non-reputation-based models

Chapter 7

Conclusion

In this dissertation, we proposed a novel hybrid, reputation-based personalised federated learning framework that can be applied in intrusion detection systems to identify real-world attacks across multiple organisations. The solution addresses several key challenges identified in the literature, including non-IID data distributions, the presence of malicious or faulty participants, scalability, and system flexibility. These challenges are mitigated through the use of hybrid topologies, reputation mechanisms, and personalisation features integrated into the framework.

The system was implemented as a network of participants using Python and Docker, enabling distributed deployment and experimentation. Established Python machine learning libraries were employed for model training, while the gRPC protocol was adopted to implement an efficient and lightweight communication layer. This design resulted in a proof-of-concept robust to evaluate the proposed contributions.

Evaluation focused on the system's ability to handle non-IID data, tolerate adversarial participants, and support model personalisation. Results showed that the system handled non-IID data effectively, though performance decreased as the number of classes in the datasets increased (achieving a mean accuracy of approximately 80%). The experiments demonstrated that both the reputation mechanism and the personalisation feature are essential for stable and accurate model training. Without personalisation, accuracy plateaued around 70%, while the absence of the reputation mechanism caused it to decline to an average of 60%. This indicates that although both features are important, the reputation mechanism is more critical and its absence more damaging. The results obtained are promising, though certain limitations remain, pointing to directions for future work, as discussed in Section 7.1.

7.1 Future Work

The proposed solution demonstrated promising results; however, several aspects remain open for improvement. From an implementation perspective, the system should be further developed to enhance both efficiency and stability beyond the proof-of-concept presented in this work. Each module and peer could be designed in a more targeted and robust manner, allowing better adaptation to the specific requirements of different organisations participating in the network. In addition,

the framework should be extended to withstand more sophisticated adversarial strategies, thereby strengthening its robustness. Another important enhancement involves supporting peer dynamics, as the current design does not yet address scenarios where peers join or leave the system during execution, a common occurrence in real-world P2P networks. Finally, the system should evolve to handle novel attacks introduced during execution more effectively, thus improving its capacity to address the non-IID challenge.

With respect to system configuration, several parameters warrant deeper exploration. Due to resource and time limitations, the experiments conducted in this dissertation were restricted to a relatively small number of participants. Future work should assess the approach in larger, more interconnected networks to validate its scalability and collaborative learning capabilities. Moreover, the number of superpeers and their maximum number of children should be revisited, as these values may be optimised depending on the overall network size. In the reputation mechanism, the threshold parameter should also be tuned under different settings to identify an optimal balance: a high threshold may restrict the system's ability to recognise new attacks, while a lower one may be overly permissive. Finally, evaluation should incorporate more advanced adversarial scenarios, moving beyond the faulty or poisoned contributions tested in this work. In particular, the system's resilience against stronger adversarial strategies, such as Byzantine attacks, should be investigated to fully validate its robustness.

Bibliography

[1] Telefonaktiebolaget LM Ericsson 1994-2025. https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/mobile-traffic-forecast, 2024.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A., G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] S. Agrawal, S. Sarkar, O. Aouedi, G. Yenduri, K. Piamrat, M. Alazab, S. Bhattacharya, P.K.R. Maddikunta, and T.R. Gadekallu. Federated learning for intrusion detection system: Concepts, challenges and future directions. *Comput. Commun.*, 195:346–361, 2022.

[4] E. Alpaydın. Introduction to machine learning. https://mitpress.mit.edu/9780262012119/introduction-to-machine-learning/, December 2021. Accessed: 2025-7-29.

[5] Z. Anastasakis, K. Psychogyios, T. Velivassaki, S. Bourou, A. Voulkidis, D. Skias, A. Gonos, and T. Zahariadis. Enhancing cyber security in IoT systems using FL-based IDS with differential privacy. In *Proceedings of the Global Information Infrastructure and Networking Symposium (GIIS)*, pages 30–34. IEEE, 2022.

[6] R. Anderson. *Security engineering: a guide to building dependable distributed systems*. Wiley, 2001.

[7] Anonymous. Cisco learning network. https://learningnetwork.cisco.com/s/question/0D53i00000KsuxDCAR/cisco-idsips-fundamentals. Accessed: 2025-9-23.

[8] Anonymous. threading — thread-based parallelism. https://docs.python.org/3/library/threading.html. Accessed: 2025-9-19.

[9] Anonymous. time — time access and conversions. https://docs.python.org/3/library/time.html. Accessed: 2025-9-19.

[10] Anonymous. Resource constraints. https://docs.docker.com/engine/containers/resource_constraints/, May 2025. Accessed: 2025-8-4.

[11] D. Bandara and A.P. Jayasumana. Collaborative applications over peer-to-peer systems–challenges and solutions. *Proceedings of the Peer Peer Netw. Appl.*, 6(3):257–276, September 2013.

[12] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, NY, 1 edition, August 2006.

[13] C. Korneck. Lightweight windows containers: Using docker process isolation in windows 10. https://poweruser.blog/lightweight-windows-containers-using-docker-process-isolation-in-windows-10-62519be76c8c, January 2019. Accessed: 2025-8-4.

[14] X. Cao, M. Fang, J. Liu, and N.Z. Gong. FLTRUST: Byzantine-robust federated learning via trust bootstrapping. *Proceedings of the 8th Annual Network and Distributed System Security Symposium (NDSS 2021)*, December 2020.

[15] J. Chen, Y. Zhao, Q. L., X. Feng, and K. Xu. FEDDEF: Defense against gradient leakage in federated Learning-Based network intrusion detection systems. *Proceedings of the IEEE Transactions on Information Forensics and Security*, 18:4561–4576, January 2023.

[16] U.B. Clinton, N. Hoque, and Khumukcham Robindro S. Classification of DDoS attack traffic on SDN network environment using deep learning. *Cybersecurity*, 7(1), August 2024.

[17] V. Danielsen. *Detecting Yo-Yo DoS attack in acontainer-based environment*. PhD thesis, Oslo Metropolitan University - OsloMet, 2021.

[18] PCI DSS. Intrusion detection / prevention. https://pcidss.com/listing-category/intrusion-detection-prevention-ids-ips/, 2024.

[19] H. Ehsan, S. Z. Dadaneh, K. Alireza, Z. Mingyuan, and Q. Xiaoning. Bayesian multi-domain learning for cancer subtype discovery from next-generation sequencing count data. *NeurIPS*, 2018.

[20] Z.A. El Houda, H. Moudoud, B. Brik, and L. Khoukhi. Securing federated learning through blockchain and explainable AI for robust intrusion detection in IoT networks. In *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2023.

[21] C. Espinosa. Signature vs. anomaly-based detection: Which is more effective? https://bluegoatcyber.com/blog/signature-vs-anomaly-based-detection-which-is-more-effective/, 2024.

[22] D. Freeze. Cybercrime to cost the world 8 trillion annually in 2023. https://cybersecurityventures.com/cybercrime-to-cost-the-world-8-trillion-annually-in-2023/, 2022.

[23] R. Hettinger G. Rossum, A. Baddeley. random — generate pseudo-random numbers. https://docs.python.org/3/library/random.html. Accessed: 2025-9-19.

[24] GeeksforGeeks. Bias-variance trade off - machine learning. https://www.geeksforgeeks.org/machine-learning/ml-bias-variance-trade-off/, February 2020. Accessed: 2025-9-28.

[25] A. Gooday. https://openmined.org/blog/federated-learning-types/. Accessed: 2025-9-30.

[26] gRPC Authors. Quick start. https://grpc.io/docs/languages/python/quickstart/. Accessed: 2025-9-23.

[27] A. Hameurlain, J. Kung, and R. Wagner, editors. *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXIII.*, volume 10430 of *Lecture Notes in Computer Science (LNCS)*. Springer, August 2017.

[28] C.R. Harris, K.J. Millman, S.J. Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, Se. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T.E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[29] J.M. Helm, A.M. Swiergosz, H.S. Haeberle, J.M. Karnuta, J.L. Schaffer, V. E. Krebs, A.I. Spitzer, and P.N. Ramkumar. Machine learning and artificial intelligence: Definitions, applications, and future directions. *Curr. Rev. Musculoskelet. Med.*, 13(1):69–76, 2020.

[30] Y. Huang, L. Chu, Z.i Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang. Personalized Cross-Silo federated learning on Non-IID data. *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI-21)*, July 2020.

[31] B. Ippolito. json — JSON encoder and decoder. https://docs.python.org/3/library/json.html. Accessed: 2025-9-19.

[32] Y. Jiang, B. Ma, X. Wang, G. Yu, P. Yu, Z. Wang, W. Ni, and R.P. Liu. Blockchained federated learning for internet of things: A comprehensive survey. *ACM Computing Surveys*, 56(10):1–37, April 2024.

[33] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *Proceedings of the ACM Trans. Inf. Syst. Secur.*, 6(4):443–471, 2003.

[34] V. Kelli, V. Argyriou, T. Lagkas, G. Fragulis, E. Grigoriou, and P. Sarigiannidis. IDS for industrial applications: A federated learning approach with active personalization. *Sensors (Basel)*, 21(20):6743, 2021.

[35] E. Kritharakis, D. Jakovetic, A. Makris, and K. Tserpes. Robust federated learning under adversarial attacks via Loss-Based client clustering. *arXiv. org*, August 2025.

[36] W. Lalouani and M. Younis. A robust distributed intrusion detection system for collusive attacks on edge of things. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1004–1009. IEEE, 2022.

[37] S. Latha and S.J. Prakash. A survey on network attacks and intrusion detection systems. In *Proceedings of the 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 1–7. IEEE, 2017.

[38] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *Proceedings of the IEEE Trans. Knowl. Data Eng.*, 35(4):3347–3366, 2023.

[39] T. Li, S.n Hu, A. Beirami, and V. Smith. Ditto: Fair and robust federated learning through personalization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6357–6368. PMLR, 18–24 Jul 2021.

[40] T. Li, A.K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. *Proceedings of the Machine Learning and Systems (MLSys) 2020*, December 2018.

[41] Z. Lu, H. Pan, Y. Dai, X. Si, and Y. Zhang. Federated learning with non-IID data: A survey. *Proceedings of the IEEE Internet Things J.*, 11(11):19188–19209, 2024.

[42] K. Lv, R. Ye, X. Huang, J. Yang, and S. Chen. Learn what you need in personalized federated learning. *Proceedings of the International Conference on Learning Representations (ICLR) 2024*, January 2024. Withdrawn.

[43] M. Martellini and A. Malizia. *Cyber and chemical, biological, radiological, nuclear, explosives challenges*. Terrorism, Security, and Computation. Springer International Publishing, 1 edition, 2017.

[44] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS 2018)*, February 2018.

[45] J. Morimoto and F. Ponton. Virtual reality in biology: could we become virtual naturalists? *Evolution (N. Y.)*, 14(1), December 2021.

[46] Nuno Neves. Mingling with the good to backdoor federated learning. *arXiv*, January 2025.

[47] T.D. Nguyen, P. Rieger, M. Miettinen, and A.R. Sadeghi. Poisoning attacks on federated learning-based IoT intrusion detection system. In *Proceedings of the Workshop on Decentralized IoT Systems and Security*. Internet Society, 2020.

[48] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[49] T. Pao and P. Wang. Nefflow based intrusion detection system. In *Proceedings of the IEEE International Conference on Networking, Sensing and Control, 2004*, volume 2, pages 731–736 Vol.2, Taipei, Taiwan, 2004. IEEE.

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Proceedings of the Journal of Machine Learning Research*, 12:2825–2830, 2011.

[51] P. Pinto. Suricata: um fantástico IDS/IPS open source. `https://pplware.sapo.pt/tutoriais/networking/suricata-um-fantastico-ids-ips-open-source/`, August 2024. Accessed: 2025-9-23.

[52] P. Pinto. Snort: a poderosa plataforma gratuita de deteção de intrusões. `https://pplware.sapo.pt/internet/snort-a-poderosa-plataforma-gratuita-de-detecao-de-intrusoes/`, January 2025. Accessed: 2025-9-23.

[53] S. K. Pye and H. Yu. Personalised federated learning: A combinational approach. *Proceedings of the 1st International Student Conference on Artificial Intelligence (STCAI'21)*, 2021.

[54] J. Rajotte, S. Mukherjee, C. Robinson, A. Ortiz, C. West, J.L. Ferres, and R.T. Ng. Reducing bias and increasing utility by federated generative modeling of medical images using a centralized adversary, January 2021.

[55] A. Ratner, P. Varma, B. Hancock, C. Ré, and other members of Hazy Lab. Weak supervision: A new programming paradigm for machine learning. `https://ai.stanford.edu/blog/weak-supervision/`, March 2019. Accessed: 2025-9-23.

[56] S. Russell and P. Norvig. *Artificial intelligence*. Pearson, Upper Saddle River, NJ, 3 edition, December 2009.

[57] V. Sajip. logging — logging facility for python. `https://docs.python.org/3/library/logging.html`. Accessed: 2025-9-19.

[58] R. Samson. Top 10 intrusion detection and prevention systems. https://www.clearnetwork.com/top-intrusion-detection-and-prevention-systems/, 2020.

[59] K. Scarfone and P. Mell. Intrusion detection and prevention systems. In *Handbook of Information and Communication Security*, pages 177–192. Springer Berlin Heidelberg, 2010.

[60] K. A. Scarfone and P. M. Mell. Guide to intrusion detection and prevention systems (IDPS). Technical report, National Institute of Standards and Technology (NIST), 2007.

[61] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings of the International Conference on Peer-to-Peer Computing*, 2002.

[62] S. Schupp. *Limitations of Network Intrusion Detection". Global Information Assurance Certification*. Global Information Assurance Certification (GIAC), 2000.

[63] SentinelOne. What is an active attack? types, detection & mitigation. https://www.sentinelone.com/cybersecurity-101/threat-intelligence/what-is-an-active-attack/, 2024.

[64] X. Shen, H. Yu, J. Buford, and M. Akon, editors. *Handbook of peer-to-peer networking*. Springer, New York, NY, 2010 edition, November 2009.

[65] K.H Shibly, M.D. Hossain, H. Inoue, Y. Taenaka, and Y. Kadobayashi. Personalized federated learning for automotive intrusion detection systems. In *Proceedings of the IEEE Future Networks World Forum (FNWF)*, pages 544–549. IEEE, 2022.

[66] S. Sinha. State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally. https://iot-analytics.com/number-connected-iot-devices/, 2024.

[67] A. Tabassum, A. Erbad, W. Lebda, A. Mohamed, and M. Guizani. FEDGAN-IDS: Privacy-preserving IDS using GAN and federated learning. *Comput. Commun.*, 192:299–310, 2022.

[68] Md. A. Talukder, Md. M. Islam, Md. A. Uddin, K.F. Hasan, S. Sharmin, S. A Alyami, and M.A. Moni. Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction. *J. Big Data*, 11(1), 2024.

[69] Y. Tan, G. Long, L. Liu, T. Zhou, Q. Lu, J. Jiang, and C. Zhang. FedProto: Federated prototype learning across heterogeneous clients. *Proceedings of the Thirty-Sixth Conference on Artificial Intelligence (AAAI-22)*, May 2021.

[70] V. Tolpegin, S. Truex, M.E. Gursoy, and L. Liu. Data poisoning attacks against federated learning systems. *Proceedings of the European Symposium on Research in Computer Security (ESORICS 2020)*, July 2020.

[71] Z. Sunny Valley. Passive attack in cybersecurity. `https://www.zenarmor.com/docs/network-security-tutorials/what-is-passive-attack`, 2024. Accessed: 2025-1-6.

[72] Q.H Vu, M. Lupu, and B.C. Ooi. *Peer-to-peer computing*. Springer, 2010 edition, 2009.

[73] M. West. Preventing system intrusions. In *Network and System Security*, pages 29–56. Elsevier, 2014.

[74] D. Wu, R. Ullah, P. Harvey, P. Kilpatrick, I. Spence, and B. Varghese. FEDADAPT: Adaptive offloading for IoT devices in federated learning. *IEEE Internet of Things Journal*, July 2021.

[75] F. Wu, L. Cui, S. Yao, and S. Yu. Inference attacks: A taxonomy, survey, and promising directions. *Proceedings of the Journal of the ACM (JACM)*, June 2024.

[76] B. Xin, W. Yang, Y. Geng, S. Chen, S. Wang, and L. Huang. Private FL-GAN: Differential privacy synthetic data generation based on federated learning. *Proceedings of the ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2927–2931, April 2020.

[77] Zenarmor. What is network intrusion? definition, detection, and prevention. `https://www.zenarmor.com/docs/network-security-tutorials/what-is-network-intrusion`, 2024. Accessed: 2025-1-6.

[78] M. Zhang, K. Sapra, S. Fidler, S. Yeung, and J.M. Alvarez. Personalized federated learning with first order model optimization. *Proceedings of the International Conference on Learning Representations (ICLR)*, December 2020.

[79] Z. Zhang, Q. Su, and X. Sun. Dim-Krum: Backdoor-Resistant federated learning for NLP with dimension-wise Krum-Based aggregation. *arXiv (Cornell University)*, January 2022.