Security of Distributed and Federated Deep Learning Systems

Duaa Salman Alqattan

School of Computing
Newcastle University

This dissertation is submitted for the degree of
Doctor of Philosophy (Integrated)

March 2025

I would like to dedicate this thesis
to my father, who passed away on February 25, 2017,
and to my father-in-law, who passed away on December 24, 2024.

Declaration

I hereby declare that this thesis is my own work unless otherwise stated. No part of this thesis has previously been submitted for a degree or any other qualification at Newcastle University or any other institution.

Duaa Salman Alqattan
March 2025

Acknowledgements

I would like to express my deepest gratitude to Dr. Varun Ojha for his continuous encouragement, guidance, and belief in my potential, which have been a constant source of motivation throughout this journey. I am also sincerely thankful to Prof. Rajiv Ranjan for his valuable guidance, attention to detail, and thoughtful feedback. Their support and clear insights have significantly shaped the quality of this thesis.

My heartfelt thanks go to my colleagues at the university who have supported and helped me in numerous ways. Their collaboration and encouragement have provided a stimulating and enjoyable environment for my research.

To my beloved husband, who has always given me a hand and stood by my side, your unwavering support and understanding have been a cornerstone of my success. Your patience and encouragement have helped me persevere through the challenges.

My three daughters (Sarah, Salma and Danah), your endless love and motivation have pushed me to strive for excellence. Your joy and inspiration have been a constant source of strength and determination for me.

To my mother, whose constant prayers have given me strength and peace, thank you for your unwavering faith in me. Your spiritual support has been a guiding light throughout this journey.

I also want to thank my sisters for their continuous support and encouragement. Your belief in my abilities and your emotional support have been a source of great motivation and comfort.

Lastly, to my cousins and friends who have always wished me the best, your positive thoughts and encouragement have been greatly appreciated. Your kindness and support have been instrumental in helping me achieve my goals.

Abstract

Distributed and federated deep learning (DL) systems, operating across the client-edge-cloud continuum, have transformed real-time data processing in critical domains like smart cities, healthcare, and industrial Internet of Things (IoT). By distributing DL training and inference tasks across multiple nodes, these systems enhance scalability, reduce latency, and improve efficiency. However, this decentralisation introduces significant security challenges, particularly concerning the availability and integrity of DL systems during training and inference. This thesis tackles these challenges through three main contributions.

- Edge-based Detection of Early-stage IoT Botnets: The first contribution involves employing Modular Neural Networks (MNN), a distributed DL approach, to develop an edge-based system for detecting early-stage IoT botnet activities and preventing DDoS attacks. By harnessing parallel computing on Multi-Access Edge Computing (MEC) servers, the system delivers rapid and accurate detection, ensuring uninterrupted service availability. This addresses the research gap in detecting early-stage IoT botnet activities as faults in network communication, enabling preventive measures before attacks escalate. Key findings include a significant reduction in false-negative rates and faster detection times (as low as 16 milliseconds), enabling early intervention in large-scale IoT environments.

- Security Assessment of Hierarchical Federated Learning (HFL): The second contribution is a security assessment of Hierarchical Federated Learning (HFL), evaluating its resilience against data and model poisoning attacks during training and adversarial data manipulation during inference. Defense mechanisms like Neural Cleanse (NC) and Adversarial Training (AT) are explored to improve model integrity in privacy-sensitive environments. This addresses the gap in systematically assessing the security vulnerabilities of HFL systems, particularly in detecting and mitigating targeted attacks in multi-level architectures. Key findings highlight that while HFL enhances scalability and recovery from untargeted attacks, it remains vulnerable to targeted backdoor attacks, especially in higher-level architectures, necessitating stronger defence mechanisms.

- Analysis of HFL Dynamics Under Attack: The third contribution examines HFL dynamics under attack using a Model Discrepancy score to analyse discrepancies in model updates. This study sheds light on the impact of adversarial attacks and data heterogeneity, providing insights for more robust aggregation methods in HFL. This addresses the gap in understanding the dynamics of HFL under adversarial attacks through model discrepancy phenomena. Key findings reveal that increased hierarchy and data heterogeneity can obscure malicious activity detection, emphasising the need for advanced aggregation methods tailored to complex, real-world scenarios.

Overall, this thesis enhances the security, availability, and integrity of Distributed and Federated DL systems by proposing novel detection and assessment methods, ultimately laying the foundation for more resilient DL-driven infrastructures.

Table of contents

List of figures

List of tables

List of publications

The contributions presented in this thesis have formed a number of research papers:

Contribution1 (Chapter 3)

This chapter is based on the paper titled "Modular neural network for Edge-based Detection of early-stage IoT Botnet". The paper was published in High-Confidence Computing.

Reference: Alqattan, D., Ojha, V., Habib, F., Noor, A., Morgan, G., and Ranjan, R. (2024). Modular neural network for Edge-based Detection of early-stage IoT Botnet. High-Confidence Computing, 100230

Contribution2 (Chapter 4)

This chapter includes research presented at the 33rd International Conference on Artificial Neural Networks (ICANN) held in Lugano, Switzerland, in September 2024. The paper, titled "Security Assessment of Hierarchical Federated Deep Learning" has contributed to ongoing discussions in Federated Learning.

Reference: Alqattan D, Sun R, Liang H, Nicosia G, Snasel V, Ranjan R, and Ojha V (2024) Security Assessment of Hierarchical Federated Deep Learning, Artificial Neural Networks and Machine Learning – ICANN 2024. Springer, Cham. Lecture Notes in Computer Science, vol 15021.

Contribution3 (Chapter 5):

This chapter is based on the paper titled "Dynamics of Hierarchical Federated Learning Under Attacks". The paper is in preparation to be submitted in High-Confidence Computing.

Reference (Under review): Alqattan D, Ranjan R, and Ojha V (2024) Dynamics of Hierarchical Federated Deep Learning Under Attacks, High-Confidence Computing. (Under review)

Chapter 1

Introduction

Distributed deep learning (DL) systems have become increasingly vital across modern industries, including smart cities, healthcare, and the industrial Internet of Things (IoT), where real-time data processing and decision-making are critical. These systems leverage a client-edge-cloud continuum, distributing computational tasks such as data collection, model training, and inference across multiple nodes. In this architecture, the client layer handles data collection, edge nodes process data closer to its source, and the cloud manages more resource-intensive tasks. This hierarchical distribution enables more responsive and robust DL applications, particularly in environments demanding real-time processing [1].

This chapter introduces the architecture of the client-edge-cloud continuum and outlines how DL processes are distributed across this framework. Following this, the research problem, key questions, aim, and objectives of this thesis are presented.

1.1 Architecture of Client-Edge-Cloud Continuum

The Client-Edge-Cloud continuum is a hierarchical architecture designed to optimise the processing and management of data across multiple layers, each with its own unique devices, roles, and responsibilities[2]. At the heart of this architecture lies the Client layer, where the user's interaction with technology begins. Devices such as smartphones, wearables, tablets, laptops, and IoT devices like smart home systems and sensors form this layer. These devices are responsible for collecting data directly from the user or the environment, and they perform basic tasks such as data preprocessing, filtering, and local decision-making. The Client layer is also where users interact with applications and services, making it crucial for responsive and efficient operation. However, these devices are often limited in computational power, storage capacity, and battery life, necessitating the offloading of more complex tasks to the next layer to ensure smooth and effective functioning[3].

Moving up the continuum, we encounter the Edge layer, which acts as an intermediary between the client devices and the cloud. This layer includes devices such as Multi-Access Edge Computing (MEC), gateways, and advanced DL-enabled hardware like smart cameras. The Edge layer is tasked with handling more demanding computational tasks that require low latency, making it essential for real-time analytics and applications where immediate processing is critical. For instance, in scenarios like autonomous driving or industrial automation, decisions must be made rapidly and locally, which the Edge layer facilitates. Additionally, this layer aggregates data from multiple client devices, ensuring that only the most relevant information is transmitted to the cloud, thus optimising bandwidth usage. By distributing the computational load, the Edge layer not only enhances the system's overall scalability but also helps in reducing the latency that would occur if all data were sent directly to the cloud for processing[4]

At the top of the continuum is the Cloud layer, which provides the most significant computational power and storage capabilities. The Cloud layer is composed of vast arrays of servers, high-performance computing clusters, and data centers. This layer is where the most complex and resource-intensive tasks are processed, such as big data analytics, DL model training, and large-scale simulations. It also serves as the central repository for storing massive amounts of data, ensuring that information is accessible and manageable over the long term. The Cloud layer supports global decision-making processes that benefit from a comprehensive view of data collected from numerous sources, enabling more informed and strategic outcomes. Moreover, the cloud hosts applications and services that need to be available worldwide, ensuring that users and devices can access necessary resources regardless of their location[2].

The need to offload tasks from one layer to another arises from the inherent limitations and strengths of each layer. Client devices, while accessible and user-friendly, are often constrained by their hardware capabilities, making it impractical to perform complex computations or store large amounts of data locally. Offloading to the Edge layer helps mitigate these issues by bringing computational resources closer to the source, thereby reducing latency and conserving bandwidth. Similarly, when tasks require even greater processing power or need to be available on a global scale, they are offloaded to the Cloud layer, which can handle such demands more efficiently[5].

Privacy and security considerations vary across these layers. The Client layer typically deals with highly sensitive personal data, making privacy a top concern. However, security at this layer can be inconsistent, often dependent on user behaviour and device capabilities. The Edge layer, being more robust, offers improved security measures, but because it aggregates data from multiple clients, it must manage privacy carefully. The Cloud layer, while providing

the most advanced security features due to significant investments by cloud providers, also carries substantial risks. Data privacy in the cloud depends on how well the data is managed and protected against unauthorised access[3].

In summary, the Client-Edge-Cloud continuum is a carefully structured hierarchical system that balances the need for computational power, latency management, and data handling across different layers. Offloading tasks between these layers ensures that each layer operates within its optimal capacity, maintaining efficiency, scalability, and security throughout the entire architecture.

1.2 Distributed DL across Client-Edge-Cloud Continuum

Distributed DL across the Client-Edge-Cloud continuum is a sophisticated approach that strategically distributes DL processes, including training and inference, across client devices, edge nodes, and cloud infrastructure[2]. Each layer in this architecture plays a distinct role: client devices handle data collection and initial processing, edge nodes manage low-latency computations and intermediate processing, and cloud servers take on intensive computational tasks and large-scale data storage. In this setup, **distributed training** involves breaking down DL model training tasks across these layers, enabling faster and more efficient learning by leveraging the computational resources at each level[6]. Similarly, **distributed inference** spreads the task of making predictions across the continuum, often positioning inference closer to the data source at the client or edge level to reduce latency and bandwidth usage[6]. The workflows for distributed training and inference can vary significantly depending on the specific requirements of the DL application. For example, an application requiring real-time decision-making, such as autonomous driving, might prioritise pushing inference tasks to the edge, where low latency is crucial[7]. In contrast, a cloud-based service that handles vast datasets might centralise training in the cloud, where extensive computational power is available[8].

The advantages of distributed DL systems are considerable. First, scalability is significantly enhanced by distributing computational tasks across multiple layers of the client-edge-cloud continuum. This enables the system to manage vast amounts of data and complex models that would overwhelm centralised systems. Second, latency is reduced by performing computations closer to the data source at the edge, a critical advantage for real-time applications such as autonomous driving and smart healthcare monitoring. Third, efficiency is improved by optimising the use of computational resources across the continuum. Localised tasks can be handled by edge nodes, reducing the need to constantly engage high-power cloud resources. Fourth, fault tolerance is increased, as the distributed nature of the system

eliminates single points of failure. If one node fails, others can take over, ensuring continuous functionality. Finally, distributed DL—particularly in privacy-centric architectures like HFL—enhances privacy protections by keeping sensitive data localised to the devices and sharing only model updates, not raw data. This feature is especially valuable in sectors such as healthcare and finance, where data privacy is paramount [1].

Figure 1.1 illustrates the architecture of distributed DL across the client-edge-cloud continuum, highlighting the varying levels of security and privacy associated with each layer.

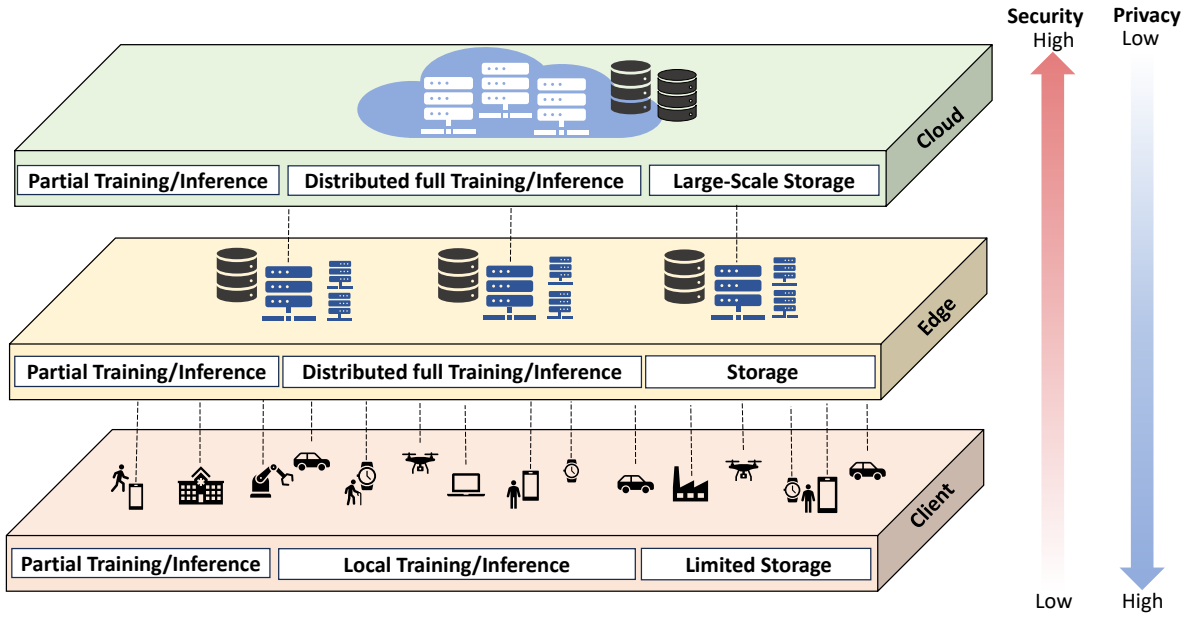

Fig. 1.1 Architecture of Distributed DL across Client-Edge-Cloud Continuum

An intriguing balance exists between the rising demand for extensive training data to refine DL models and the practice of limiting data usage to ensure privacy protection[9]. Therefore, Distributed DL can also be classified based on data sharing and privacy needs into two categories: **Data-Centric Distributed DL** and **Privacy-Centric Distributed DL**. **Data-Centric Distributed DL** focuses on the efficient sharing and processing of large datasets across the continuum, optimising performance and accuracy by fully utilising available data. A typical example of this is a smart city application that collects data from various IoT sensors throughout the city. This data, including traffic patterns, environmental conditions, and public safety metrics, is streamed to edge nodes for initial processing, but the bulk of data processing and DL model training happens in the cloud. Here, the cloud's vast computational resources analyse patterns, optimise traffic flow, or predict environmental changes. This approach maximises data usage to improve model accuracy and performance, but it often

comes at the cost of increased data sharing across the network, which can pose privacy risks[5].

In contrast, **Privacy-Centric Distributed DL** priorities safeguarding sensitive information by limiting data movement and using techniques like federated learning, where models are trained locally on client or edge devices without sharing the raw data. This approach is particularly crucial in domains where data privacy and security are paramount, such as healthcare or finance. For instance, in a healthcare application using federated learning, individual hospitals or medical devices train local DL models with their own patient data. These models are then aggregated at the edge or cloud level without the raw data ever leaving the local devices. This method ensures that sensitive patient information remains private while still benefiting from collective DL model improvements[10].

The key difference between Data-Centric and Privacy-Centric Distributed DL lies in how they balance performance and data privacy. **Data-Centric Distributed DL** tends to prioritise performance by enabling extensive data sharing and centralised processing, in the distributed edge or cloud nodes, to achieve the highest possible accuracy and efficiency. However, this comes with the trade-off of potential privacy risks, as sensitive data is more widely distributed and potentially exposed[5]. On the other hand, **Privacy-Centric Distributed DL** focuses on protecting data privacy by limiting data movement and employing federated learning, where only model updates are shared rather than raw data. This approach is essential in situations where data sensitivity is critical. The trade-off here is that it may result in slightly lower performance, as DL models might not access as much diverse data for training, and the distributed nature of the process might be less efficient compared to data sharing and centralised processing in Data-Centric approach [10].

The choice between Data-Centric and Privacy-Centric Distributed DL approaches involves a trade-off between maximising DL performance and ensuring data privacy. Data-Centric approaches excel in scenarios where data availability and model accuracy are critical, while Privacy-Centric approaches are better suited for applications where safeguarding sensitive information is the top priority. Next, the details of distributed training and distributed inference under these two approaches will be explained to further clarify how each approach functions in practice.

1.2.1 Data-Centric Distributed DL

Distributed Training

In Data-Centric Distributed DL, distributed training is a crucial process where training tasks are strategically distributed across clients, edge nodes, and cloud servers. This approach

enables DL models to be trained across multiple nodes that are often located in different geographical locations, allowing for a more efficient and scalable training process. By leveraging the strengths of each layer in the Client-Edge-Cloud continuum, Distributed DL systems can effectively manage the complexities of large-scale model training[11].

The process begins with distributed data collection from client devices, which send raw data to nearby edge nodes for initial processing. These edge nodes act as intermediaries, reducing the amount of data that needs to be transmitted to the cloud and enabling more responsive processing. Once the data is collected and preprocessed at the edge, parallelism technologies are implemented to train the DL model. The main parallelism technologies used in distributed training include Data Parallelism, Model Parallelism, Hybrid Parallelism, Ensemble Parallelism and Expert Parallelism. These technologies can be applied solely at the edge or distributed between edge nodes and cloud servers, depending on the specific requirements and constraints of the DL application.[12].

1. Data Parallelism involves distributing different portions of the dataset to various edge nodes. Once the data is sent from the clients to the edge, the edge nodes train identical copies of the model on different subsets of the data. These edge nodes process their assigned data in parallel and train the same model. After training, the updates (such as gradients) are aggregated at a higher layer, such as the cloud, to update the global model. This method leverages the computational power of the edge while ensuring that each edge node contributes to the training of the same model. In this way, Data Parallelism helps scale up the training process without overloading any single node, optimising training speed while ensuring the model sees diverse data from different clients[13]

2. Model Parallelism involves splitting the model itself across different nodes (or GPU). Clients send their raw data to their respective edge nodes, where different parts of the model are trained. Each edge node handles a specific section of the model, such as the early layers dealing with feature extraction, while more complex, computationally heavy layers might be trained at the cloud level. As data flows from the edge nodes to the cloud, each section of the model is trained on the respective node responsible for that part. This division of tasks allows for training very large models that might not fit on a single node, spreading the computational requirements across the continuum. Model Parallelism is particularly advantageous for large-scale DL systems requiring distributed resources for efficient model training [14].

3. Hybrid Parallelism combines the principles of Data Parallelism and Model Parallelism. In this approach, the clients send data to their associated edge nodes, where training occurs. In Hybrid Parallelism, each edge node not only processes different subsets of the data (Data Parallelism) but also handles different parts of the model (Model Parallelism). For example, while data from various clients is distributed across edge nodes, the model may be partitioned between the edge and the cloud, with the cloud processing the more complex sections of the model. This method maximises resource utilisation by distributing both data and model workloads across the client-edge-cloud continuum, balancing computational load and ensuring both scalability and efficiency in model training.[15].

4. Ensemble Parallelism involves training multiple distinct models on different subsets of data or different data distributions, with each model being trained on edge nodes after receiving data from clients. Once the models are trained, their outputs are combined at the cloud or edge to produce a final decision. In Data-Centric Distributed DL, clients send their raw data to their nearest edge nodes, where each edge node may train a different model on its own subset of the data. The cloud or a central edge node aggregates the predictions from these independent models to generate more robust and accurate decisions. This approach benefits from having diverse models trained on varied data, enhancing the overall generalisation and robustness of the system[16].

5. Expert Parallelism divides the DL task into smaller, specialised sub-tasks, with each module (expert) focusing on a specific part of the problem. Clients send their data to edge nodes, where each edge node acts as an expert specialised in solving a specific part of the problem. Once training is completed, the outputs from these edge-based expert models are aggregated at a higher layer, such as another edge node or the cloud, to produce a final decision. This approach not only speeds up training by enabling parallelisation but also enhances model accuracy by leveraging the specialisation of each expert module. Expert Parallelism can be implemented through Modular Neural Networks (MNN), which is particularly well-suited for distributed environments like smart cities, where different types of data need to be processed and analysed in parallel across multiple edge nodes[17].

These Parallelism training mechanisms in Data-Centric Distributed DL are illustrated in Figure 1.2

Distributed Inference

Distributed inference within Data-Centric Distributed DL addresses the challenges associated with transmitting raw data to remote servers for deep learning (DL) inference. This traditional

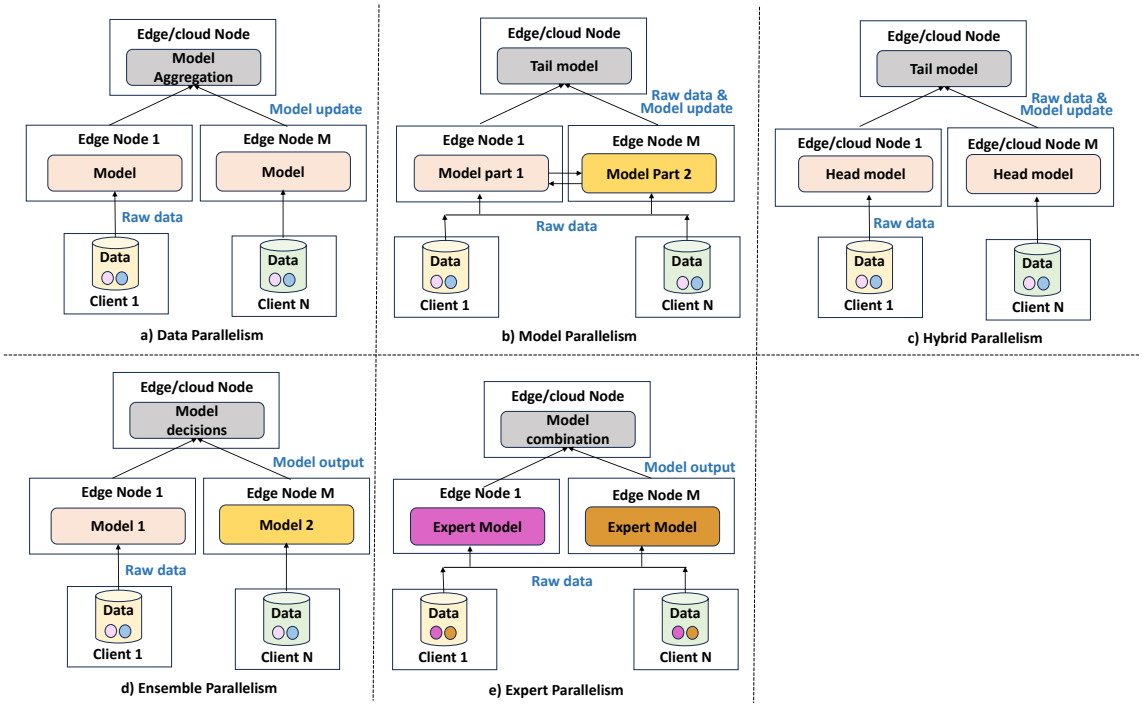

Fig. 1.2 Data-Centric Distributed Training mechanisms

approach can result in significant communication overhead, leading to increased latency and reduced energy efficiency, particularly in environments where timely responses are critical. To overcome these limitations, DL models can be segmented into multiple parts and allocated across different computational nodes or GPUs, thereby optimising the inference process[2].

Under Data-Centric Distributed DL, two primary techniques for distributed inference are commonly employed: **Edge-only Inference** and **Edge-Cloud Inference**, as illustrated in Figure 1.3.

1) Edge-only Inference: In the edge inference technique, client devices transmit data to a nearby edge server, which then performs the inference using a DL model that has been partitioned across multiple edge nodes. The model is often divided horizontally, meaning that different layers of the model are processed at different edge nodes[2]. This approach significantly reduces the communication overhead that would otherwise occur if the raw data were sent to a distant cloud server for processing. Furthermore, edge inference can be enhanced through techniques like ensemble inference[16], where multiple models are used in parallel to improve the confidence and accuracy of the results, and modular neural networks[17], which allow for more robust inference outputs. Once the edge server completes the inference, the results are sent back to the client devices, providing a quick and efficient response.

2) Edge-Cloud Inference: The edge-cloud inference approach leverages the combined computational power of both the edge and the cloud, creating a synergy that optimises the inference process. In this mode, clients are responsible for collecting the input data, which is then initially processed at the edge. The edge node evaluates whether it has sufficient resources and model capacity to handle the inference task. If the edge node determines that the DL model deployed locally can successfully process the input data, it completes the inference and returns the results to the client. However, if the edge node lacks the necessary computational power or if the required model complexity exceeds what is available at the edge, the remaining inference task is offloaded to a more powerful DL model deployed in the cloud[18].

A key feature of edge-cloud inference is model early exiting[19], a technique that allows the inference process to stop at an earlier stage within the distributed computing hierarchy if the target accuracy threshold is met. For example, in a deep neural network (DNN) with multiple output nodes, the inference can conclude at an intermediate layer if the desired accuracy is achieved, reducing the need for further processing and thus saving resources. If the accuracy requirement is not met at the edge, the remaining portions of the model are executed in the cloud, ensuring that the final result meets the necessary standards. This joint inference approach minimises both communication overhead and resource usage, making it a highly efficient method for scenarios where computational demands vary[2].

Figure 1.3 illustrates Data-Centric Distributed inference.

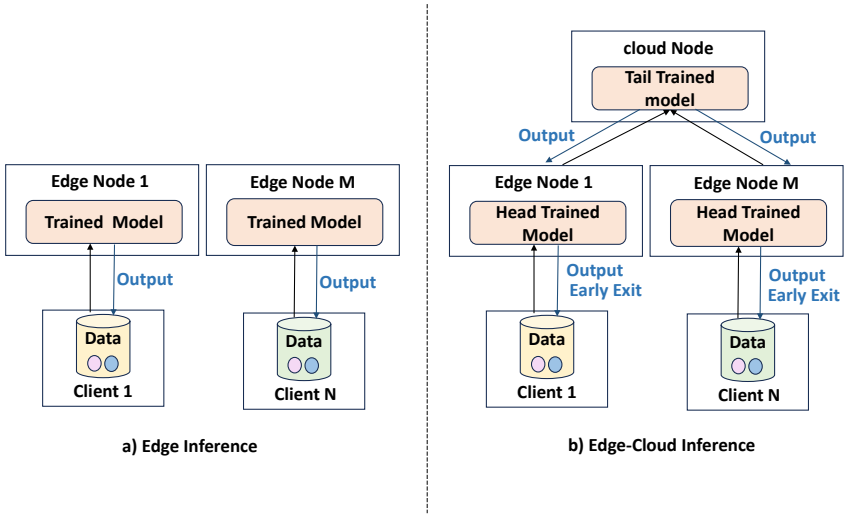

Fig. 1.3 Data-Centric Distributed Inference mechanisms

1.2.2 Privacy-Centric Distributed DL: Hierarchical Federated Deep Learning

Distributed Training

In Privacy-Centric Distributed DL, distributed training remains a vital process, much like in Data-Centric Distributed DL, where training tasks are strategically distributed across clients, edge nodes, and cloud servers. However, the key difference in Privacy-Centric Distributed DL is the focus on safeguarding sensitive data. The training process is designed to ensure that data remains secure and private, minimising the risk of exposure by reducing the movement of raw data across the network[6]. To achieve this level of privacy, **Federated Deep Learning** is employed as the primary method for distributed training. When applied across the Client-Edge-Cloud continuum, this approach is referred to as **Hierarchical Federated Deep Learning**. This method is similar to the Data Parallelism approach, specifically tailored to maintain data privacy by allowing training to occur locally on client devices, with only model updates being shared rather than the raw data itself[20].

Hierarchical Federated Deep Learning operates by enabling local training on client devices or edge nodes, where each device uses its own data to update a local copy of the DL model. Instead of transmitting the raw data to a central server, each device sends model updates, such as gradients or weights, to an aggregate server. The server, which could be an edge node or a cloud server, then performs a process known as Federated Averaging. In this process, the server averages the model updates received from multiple nodes to create a global model that reflects the collective learning of all participating nodes[21].

This hierarchical approach can be implemented in several ways, depending on the complexity and scale of the application. For example, in a three-level hierarchy (client-edge-cloud), client devices perform local training, edge servers aggregate the updates from clients within their proximity, and the cloud server further aggregates updates from multiple edge servers to create a global model[22]. Alternatively, a four-level hierarchy might be employed, adding a regional or fog server between the edge and the cloud. In this setup, regional servers aggregate updates from multiple edge nodes before sending the combined updates to the cloud, which then produces the final global model. This multi-level aggregation further enhances scalability and reduces the computational load on any single server[23]. Hierarchical Federated Deep Learning is particularly useful in applications where data privacy is paramount, such as in healthcare, finance, or personalized services. For instance, in a healthcare application, patient data could be used to train a predictive model for diagnosing diseases. With hierarchical federated learning, the sensitive medical data never leaves the hospital's servers or patient's devices. Instead, only model updates are shared, ensuring that

patient privacy is maintained while still benefiting from the collective learning of models trained across multiple hospitals or regions[24].

Splitting Learning is another technique that complements Privacy-Centric Distributed DL[25]. In this approach, a deep learning model is divided into segments, where the initial segments are trained on clients, and the subsequent segments are processed on more powerful servers, such as those in the cloud. This method ensures that only intermediate model activations, not raw data, are transmitted to the cloud, thus preserving privacy while enabling the training of complex models that require more computational resources than what is available on local devices. Splitting Learning is particularly useful in cases where the model is too large to be entirely trained on clients or when the computational tasks need to be distributed for efficiency[26].

Privacy-Centric Distributed DL, utilising Hierarchical Federated Deep Learning and Splitting Learning (Figure 1.4), presents a powerful framework for distributed training that emphasises data privacy while supporting scalable and efficient DL model development. This method is particularly advantageous in scenarios where ensuring data security is crucial, offering a versatile and effective solution for deploying DL across the Client-Edge-Cloud continuum. The primary focus of this thesis is on Hierarchical Federated Deep Learning.

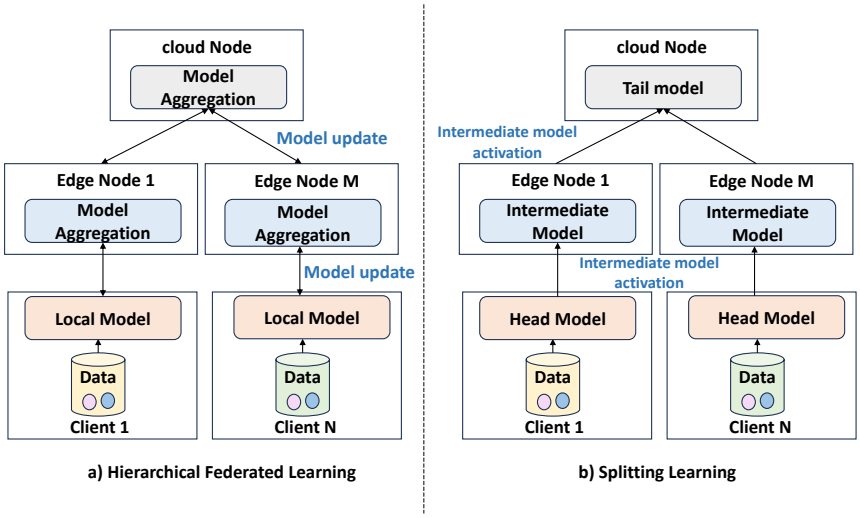

Fig. 1.4 Privacy-Centric Distributed Training mechanisms

Distributed Inference

In Privacy-Centric Distributed DL, distributed inference is carefully designed to prioritise data privacy by beginning the inference process at the client level, where the data is initially generated. By keeping the data at the client as long as possible and only transmitting

intermediate result to higher layers of the continuum (such as edge or cloud), this approach ensures that sensitive information remains protected. This method reduces the exposure of raw data, minimising privacy risks while still enabling the use of advanced DL models. Figure 1.5 illustrates Privacy-Centric Distributed inference mechanisms.

1) Client-only Inference: Given the challenges of communication latency and privacy concerns, client devices take full responsibility for performing inference using the deployed DL models. In this scenario, the entire deep learning (DL) model is executed locally on the client device, ensuring that raw data never leaves the device. This approach is highly privacy-centric, as it keeps the data confined to the client. However, it requires that the model be optimised to run efficiently on resource-constrained devices, such as smartphones or IoT devices, which may not have the computational power to handle large models[2].

2) Client-Edge Inference: In the client-edge inference mode, the client device begins the inference process by executing part of the DL model locally, according to its computational capabilities. The intermediate results, rather than the raw data, are offloaded to an edge server for further processing. The edge server completes the remaining inference tasks and sends the final results back to the client device. This mode offers greater flexibility compared to client-only inference, as it leverages distributed resources to perform computation, thus balancing the load between the client and the edge. It allows for more complex models to be used while still maintaining a focus on privacy by limiting the data that is transmitted[2].

3) Client-Edge-Cloud Inference: In the client-edge-cloud inference mode, the inference process starts at the client device, where the initial data is processed and the first part of the DL model is executed. If the client device lacks the necessary computational power or if the model is too complex, the intermediate data is then offloaded to the edge server. The edge server continues the inference by processing additional layers of the model. If further computational resources are needed or if the model requires more processing power than the edge can provide, the remaining inference tasks are offloaded to the cloud server. The cloud, with its vast resources, completes the inference and returns the final result down the continuum—first to the edge, and then back to the client device[6].

A significant enhancement in the client-edge-cloud inference mode is the use of the early exit technique as discussed earlier for edge-cloud inference in data-centric Distributed DL. As the inference progresses from the client to the edge and possibly to the cloud, the model can "exit" at any of these nodes if the desired accuracy threshold is met. This approach not only reduces the computational burden by avoiding unnecessary processing of deeper layers

but also minimises communication overhead and latency. If the early exit conditions are satisfied at the client or edge level, the inference can terminate without the need to involve the cloud, thus preserving privacy and optimising resource usage. If the required accuracy is not achieved early, the inference task continues up the hierarchy, ensuring that the final result meets the application's accuracy requirements[27].

In Privacy-Centric Distributed DL, DL inference at the client layer garners significant attention due to the privacy benefits it offers. However, client devices are typically limited in resources, while DL models, which often consist of millions of parameters, require high-performance computing units such as CPUs, GPUs, and DSPs[2]. To address this challenge, it is essential to optimise DL models through techniques like model compression [28]. These techniques help to reduce the computational load, allowing the models to run on less powerful devices while maintaining a balance between resource consumption and model performance. By optimising the models, it's possible to achieve effective inference on resource-constrained devices without compromising on privacy or significantly degrading performance[29].

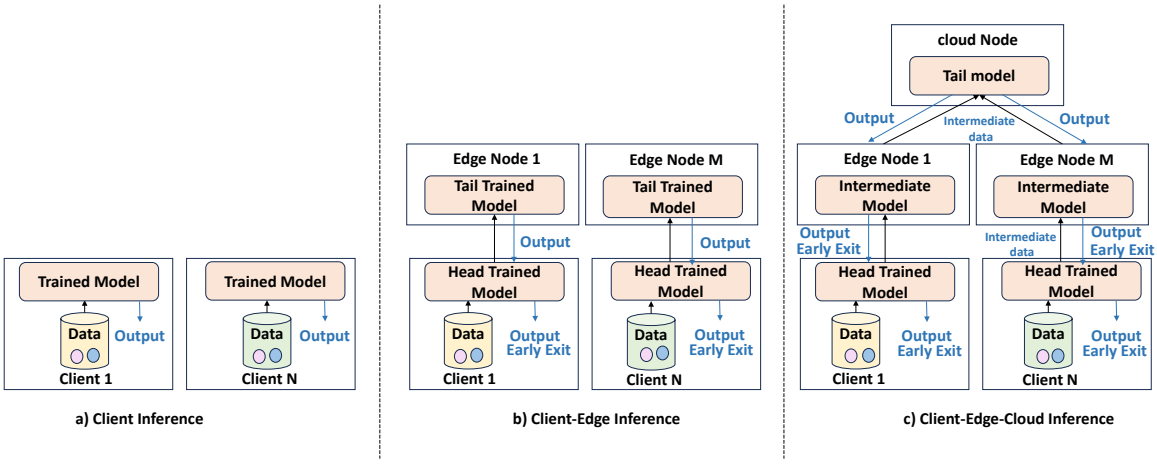

Fig. 1.5 Privacy-Centric Distributed Inference mechanisms

1.3 Problem Definition

Despite these benefits, the decentralisation of distributed DL systems introduces substantial security challenges, particularly concerning system availability and model integrity. The client layer, which includes potentially insecure devices such as IoT sensors, is vulnerable to botnet attacks that can launch Distributed Denial of Service (DDoS) attacks, disrupting essential services in smart cities and other critical infrastructures. Early detection of botnet activities is essential to maintaining system availability. Furthermore, adversarial attacks

pose a threat to the integrity of distributed DL models, where data and model poisoning can corrupt both the data and models during training and inference, leading to inaccurate and potentially harmful outcomes. This is especially concerning in privacy-centric architectures like HFL, where insecure devices at the client layer participate in both training and inference processes. Figure 1.1 highlights the varying levels of security and privacy associated with each layer of the architecture of distributed DL across the client-edge-cloud continuum.

One of the most pressing challenges is the detection of early-stage IoT botnet activities. IoT devices, widely deployed in smart city infrastructures, are vulnerable to botnets that can launch Distributed Denial of Service (DDoS) attacks, causing widespread service outages. These attacks can severely disrupt real-time applications that depend on uninterrupted service, such as traffic management and public safety systems. Existing detection methods often fall short due to their inability to identify botnet activities in their nascent stages, leading to significant detection delays and high false-negative rates. Thus, there is a need for a more efficient detection system that is also suitable for the distributed nature of DL systems, allowing it to leverage the computational resources across the client-edge-cloud continuum for faster and more accurate identification of threats before they escalate [30].

In addition to availability concerns, the integrity of DL models in Hierarchical Federated Learning (HFL) is at risk from data and model poisoning attacks. These attacks can be initiated during both the training and inference phases and can corrupt DL models, leading to inaccurate or biased predictions. Given that HFL is often used in privacy-sensitive applications like healthcare and finance, these integrity threats pose serious risks to the safety and reliability of DL systems. Moreover, the hierarchical structure of HFL introduces additional complexity in defending against such attacks, particularly when adversarial behaviours are distributed across different layers of the client-edge-cloud continuum [31].

Another significant challenge is the lack of understanding of HFL dynamics under adversarial attacks. While traditional security assessments focus on measuring the performance of DL models under attack, there is a need for deeper insights into how model discrepancies evolve over time during attacks. Such an understanding is critical for developing more robust defense mechanisms that can adapt to dynamic adversarial environments and ensure long-term model integrity.

The interrelation between IoT botnet detection and HFL security lies in their shared goal of improving the security of distributed DL systems. IoT botnets, if undetected, can severely disrupt the client-edge-cloud continuum, impacting the availability of DL systems. On the other hand, the hierarchical structure of HFL introduces additional vulnerabilities that adversaries can exploit to compromise the integrity of DL models. Both challenges threaten

critical aspects of distributed DL systems—availability in data-centric systems and integrity in privacy-centric systems.

These challenges were chosen as primary focus areas because they highlight a broad spectrum of vulnerabilities in distributed DL systems, reflecting the diversity of architectures and applications in this field. By addressing the availability threats posed by IoT botnets in data-centric distributed DL systems and the integrity risks in privacy-centric HFL, the thesis underscores the importance of adopting comprehensive security strategies that cater to the specific needs of these two domains. This approach ensures a holistic understanding and mitigation of the security challenges facing modern distributed DL systems.

Overall, the central problem addressed in this thesis is the development of security solutions that can enhance both the availability and integrity of Distributed DL systems. This includes designing an efficient detection system suitable for distributed environments, capable of addressing IoT botnet threats, and conducting comprehensive security assessments of HFL systems under adversarial attack, with a focus on understanding the dynamics of model discrepancies in Distributed DL frameworks.

1.4 Research Questions

Building upon the challenges identified in the Problem Definition, this thesis focuses on addressing critical vulnerabilities in distributed DL systems, particularly those impacting system availability and model integrity. To explore these issues, the following research questions have been formulated and serve as the foundation for the studies presented in Chapters 3, 4, and 5.

Research Questions1:

How can early-stage IoT botnets be effectively detected at the edge to prevent Distributed Denial of Service (DDoS) attacks in data-centric distributed DL systems?

This research delves into two key sub-questions. The first explores how early-stage IoT botnet behaviours, such as scanning and command-and-control (CnC) phases, can be modelled as faults in network communication within smart city systems (Research Questions 1.1). The The second investigates how a distributed DL model can be designed to accurately detect these botnet-induced faults in real-time (Research Questions 1.2). Key considerations include optimising the model to minimise false-negative rates while maintaining low detection latency. Additionally, the research aims to leverage the distributed nature of the client-edge-cloud continuum, utilising parallel processing (MNN) at the edge to enhance detection speed and system availability.

Research Questions2:

How does the HFL architecture impact the robustness of HFL systems against adversarial attacks and what defense mechanisms can enhance their security?

This question explores how adversarial attacks, specifically data poisoning and model poisoning, compromise the accuracy and reliability of HFL models during the training phase. It also evaluates how inference-time attacks, such as adversarial data manipulation, affect the robustness of HFL systems. The study explores the effectiveness of defense mechanisms, such as Neural Cleanse (NC) and Adversarial Training (AT), in mitigating the impact of these attacks across various levels of the client-edge-cloud continuum. The goal is to assess how the architecture of HFL influences its robustness compared to traditional FL systems in privacy-sensitive applications.

Research Questions3:

How do different attack scenarios impact the dynamics of Hierarchical Federated Learning (HFL) systems, and what factors influence model discrepancies during these attacks?

This question focuses on understanding the dynamics of HFL under various attack scenarios. it develops a Model Discrepancy score to measure and analyse discrepancies in model updates within HFL architectures under adversarial attack. The study examines how different factors—such as data heterogeneity, attack sophistication, and the number of aggregation layers—affect model discrepancies. The temporal, spatial, and clustering dynamics of HFL systems under attack are analysed to provide insights into how adversarial behaviour evolves.

1.5 Objectives and Contributions

This thesis makes significant contributions to the field of Distributed DL security, particularly in enhancing the availability and integrity of systems operating across the client-edge-cloud continuum. These contributions are aligned with three main objectives:

Objective 1: Development of Modular Neural Network for Edge-Based Detection of Early-Stage IoT Botnets

The first objective is to develop a Modular Neural Network (MNN) system for edge-based detection of early-stage IoT botnets. This system aims to improve the availability of data-centric distributed DL systems by preventing Distributed Denial of Service (DDoS) attacks at the edge.

Contribution: This objective is achieved through the development of an edge-based detection system, EDIT (Edge-based Detection of Early-Stage IoT Botnets). EDIT models early-stage IoT botnet activities, such as scanning and command-and-control (CnC) phases, as faults in network communication. By leveraging MNN, the detection process is broken

into expert binary classification tasks, which can be executed in parallel on Multi-Access Edge Computing (MEC) servers. This approach ensures fast, accurate detection with a low false-negative rate, preventing disruptions to system availability.

Objective 2: Assessment of Security Vulnerabilities of Hierarchical Federated Deep Learning (HFL)

The second objective is to conduct a thorough security assessment of Hierarchical Federated Learning (HFL) systems, focusing on their resilience against adversarial attacks during both training and inference stages. This research addresses the integrity of privacy-centric distributed DL systems.

Contribution: This objective is addressed by evaluating the impact of data poisoning and model poisoning attacks on HFL systems during training, as well as adversarial data manipulation during inference. Defense mechanisms such as Neural Cleanse (NC) and Adversarial Training (AT) are implemented and tested to enhance the robustness of HFL models. The assessment highlights HFL's hierarchical structure's advantages in mitigating some attacks while identifying areas where vulnerabilities persist.

Objective 3: Understanding the Dynamics of Hierarchical Federated Deep Learning Under Attacks

The third objective is to explore the dynamics of hierarchical Federated Learning (HFL) under adversarial attacks, with a focus on analysing model discrepancies during training. This research provides insights into how different attack scenarios affect training behaviour and model updates in HFL systems, especially in environments with high data heterogeneity.

Contribution: This objective is achieved by developing a Model Discrepancy Score that integrates multiple metrics—Dissimilarity, Distance, Uncorrelation, and Divergence—to detect and analyse discrepancies in model updates. A comprehensive experimental study comparing 3-level and 4-level HFL architectures under various attack scenarios reveals how adversarial attacks affect HFL dynamics. These findings highlight the limitations of existing aggregation methods and suggest directions for improving HFL robustness.

1.6 Thesis Structure

The rest of this thesis is structured as follows:

Chapter 2: Background and Literature Review This chapter provides a comprehensive review of security threats in Distributed DL systems and examines existing defense mechanisms. It identifies gaps in current research and situates the thesis within the broader context of Distributed DL security.

Chapter 3: Modular Neural Network for Edge-Based Detection of Early-Stage IoT Botnets This chapter presents the development of EDIT, an edge-based detection system for early-stage IoT botnets, using Modular Neural Networks (MNN). It details the design, implementation, and experimental evaluation of EDIT, demonstrating its effectiveness in detecting botnet activities with high accuracy and low latency.

Chapter 4: Security Assessment of Hierarchical Federated Learning (HFL) Under Adversarial Attacks This chapter explores the security vulnerabilities of HFL systems under adversarial attacks. It evaluates the impact of data and model poisoning during training and adversarial data manipulation during inference. The chapter also examines defense mechanisms, such as Neural Cleanse and Adversarial Training, to improve HFL's robustness.

Chapter 5: Dynamics of Hierarchical Federated Learning Under Attacks This chapter investigates the dynamics of HFL systems under various attack scenarios by introducing the Model Discrepancy Score. It provides an in-depth analysis of discrepancies in model updates and compares the performance of 3-level and 4-level HFL architectures, offering insights into improving aggregation methods and system robustness.

Chapter 6: Conclusions and Future Work The final chapter summarises the key contributions of the thesis, addressing the research questions and objectives. It reflects on the limitations of the study and proposes directions for future research, including advancements in IoT botnet detection and further strengthening of HFL security.

Chapter 2

Background and Literature Review

This chapter provides an in-depth exploration of the security challenges and defense mechanisms in Distributed DL systems operating across the client-edge-cloud continuum. It also positions the thesis within the existing body of research by identifying key gaps and contributions. To support this discussion, Table 2.1 presents a concise summary of security threats and assesment, corresponding Existing Proposed Ap proaches, research gaps, and the thesis approach to address the gap.

Table 2.1 Summary of Security Threats/Assesment and Proposed Approaches in Distributed DL

Security Threats/Assesment	**Threats/Assesment Description**	**Existing Proposed Approaches**	**Approach Type**	**Key References**	**Research Gap**	**Thesis Approach to Address the Gap**
DoS Attacks and IoT Botnet Attacks	Overloading servers with excessive traffic or launching coordinated attacks using compromised IoT devices	DoS and Botnet Detection	Defense Mechanism	[32–36]	Limited research leveraging parallelism in distributed DL for detecting DoS and botnet attacks. Need for distributed, accurate, and rapid DL-based detection methods capable of identifying botnets in early stages of formation.	Develop edge-based detection systems using parallel Modular Neural Networks (MNN) for real-time mitigation. Introduce fault-based modelling of early-stage botnet behaviours for rapid and accurate detection.
Adversarial Attacks	Crafting adversarial inputs to mislead models	Adversarial Training	Defense Mechanism	[37–40]	Adversarial Training has not yet been explored within the framework of hierarchical federated learning. Although these studies provide robust aggregation methods for hierarchical federated learning, there is still a need to improve their effectiveness.	Evaluate adversarial training techniques within hierarchical federated learning for improved security.
Data and Model Poisoning	Injecting malicious data or manipulating model updates to corrupt models	Model Discrepancy-based Robust Aggregation Methods	Defense Mechanism	[41, 23, 42]	The model discrepancy detection primarily relies on a single metric, cosine similarity. To enhance these methods, further analysis of their efficiency in hierarchical federated learning is necessary, along with the exploration of additional model discrepancy metrics.	Implement robust aggregation methods and model discrepancy scoring tailored to non-IID environments, while incorporating multiple metrics beyond cosine similarity.
Data and Model Poisoning	Injecting malicious data or manipulating model updates to corrupt models	Model Pruning and Cleaning	Defense Mechanism	[43–46]	Despite the successful application of model pruning in traditional federated learning, there remains a research gap in studying the efficiency of model pruning within distributed deep learning mechanisms, including hierarchical federated learning.	Evaluate model pruning techniques within hierarchical federated learning for improved security.
System Performance	Evaluating the effectiveness of systems under attack scenarios	Performance Under Attack and Defense	Security Assessment	[47–51]	Limited studies systematically benchmarking the performance of distributed DL systems , particularly hierarchical federated learning, under attack and defense conditions.	Develop a comprehensive framework to evaluate the performance of distributed DL systems under both attack and defense conditions.
System Behaviour	Observing the impact of attacks on system dynamics and how defense mechanisms mitigate these impacts	System Dynamics Under Attack and Defense	Security Assessment	[47–51]	Insufficient analysis of how attack influence hierarchical federated learning system behaviour over time.	Propose a analysis framework to study how attacks impact hierarchical federated learning dynamic, with insights into temporal and spatial effects.

2.1 Security Threats in Distributed DL

Distributed DL across the Client-Edge-Cloud Continuum introduces a range of security challenges that differ significantly from those found in traditional centralised DL systems. These challenges arise primarily due to the expanded attack surface that comes with distributing DL processes—both training and inference—across multiple layers, including clients, edge nodes, and cloud servers. Unlike centralised systems, where security can be more easily managed within a controlled environment, Distributed DL requires careful consideration of the specific security threats associated with each layer[2].

In centralised DL systems, both training and inference typically occur within a secure, controlled environment such as a cloud server or a data center. This centralised approach confines the attack surface to a single location, making it easier to implement robust security measures like firewalls, encryption, and access controls [52]. However, Distributed DL involves spreading these processes across multiple layers, significantly broadening the attack surface. For distributed training, data and model updates are shared between clients, edge nodes, and cloud servers, creating multiple potential entry points for attacks. Similarly, distributed inference, which involves performing inference tasks across the client, edge, and cloud, exposes the system to new vulnerabilities as data and intermediate results are transferred between these layers. The expanded attack surface in Distributed DL systems introduces new risks, particularly in environments where devices and networks may be less secure than centralised data centers[53].

To secure Distributed DL, The CIA triad security requirements—confidentiality, integrity, and availability—must be addressed. Confidentiality refers to the protection of data and models from unauthorised access and disclosure. Confidentiality is critical for both training and inference, as sensitive data and model parameters are shared across multiple nodes, increasing the potential for exposure. Integrity involves ensuring that data and model updates are accurate and have not been tampered. This is essential in distributed training, where corrupted data or models can lead to faulty DL systems, and in inference, where incorrect results can have serious consequences. Availability refers to the requirement that DL services remain accessible and functional, even in the face of attacks. Distributed DL systems, particularly those involving real-time or mission-critical application[54], must be resilient against disruptions such as Denial of Service (DoS) attacks. Ensuring these three pillars of security is vital for the safe and effective operation of Distributed DL systems[55]. This thesis focuses on the security aspects of availability and integrity. Consequently, this chapter will primarily explore the threats to these two principles, as well as the corresponding defense mechanisms.

2.1.1 Availability Threat Model

Availability threats in Distributed DL refer to attacks that disrupt the operation of DL services, making them inaccessible or non-functional. These threats are particularly concerning because they can severely impact both the training and inference processes by interrupting the communication and computational tasks that are essential for the system's operation[56].

Denial of Service (DoS) attack

One of the most prominent availability threats is the Denial of Service (DoS) attack, where attackers flood servers—whether at the edge or in the cloud—with excessive traffic[57]. This overwhelming surge of traffic can cause the servers to become unresponsive, rendering DL services (training and inference) unavailable to the clients. DoS attacks pose a significant risk to distributed systems because both training and inference rely on continuous and reliable communication between the various layers of the Client-Edge-Cloud Continuum[58].

Attacker's Knowledge: The effectiveness of a DoS attack depends largely on the attacker's understanding of the target system. In a white-box scenario, the attacker possesses detailed knowledge of the system's architecture, including the specific servers responsible for critical operations like model training or inference. This allows the attacker to focus the flood of traffic on key components of the system, maximising the impact. In a black-box scenario, the attacker lacks detailed knowledge but can still identify target servers by observing IP addresses and response patterns. Even with limited information, an attacker can execute a DoS attack by directing massive traffic to visible, essential servers, disrupting their operations[59].

Attacker's Goal: The primary objective of a DoS attack is to overwhelm the computational and network resources of the target servers, causing them to become unresponsive or crash. By disrupting the availability of servers, the attacker prevents the distributed DL system from performing critical functions such as training and inference. This leads to system outages, degraded performance, and potential safety risks, especially in applications requiring real-time processing (e.g., autonomous vehicles, healthcare systems). The motivations behind a DoS attack can range from extortion and reputational damage to creating vulnerabilities for future attacks.

Attacker's Ability: An attacker's ability in a Denial of Service (DoS) attack is primarily determined by their capacity to generate an overwhelming volume of traffic from a single origin or a few controlled systems. The attacker exploits the system's vulnerabilities, such as unoptimized handling of incoming traffic or insufficient capacity, to cause service disruptions. The attacker may also leverage certain software vulnerabilities to crash or freeze the server. A key factor in the effectiveness of a DoS attack is the attacker's ability to maintain a sustained

overload, exhausting the target system's resources and rendering its services unavailable to clients.[60].

Attacker's Strategies: Attackers use several strategies to execute DoS attacks against DL services (figure 2.1):

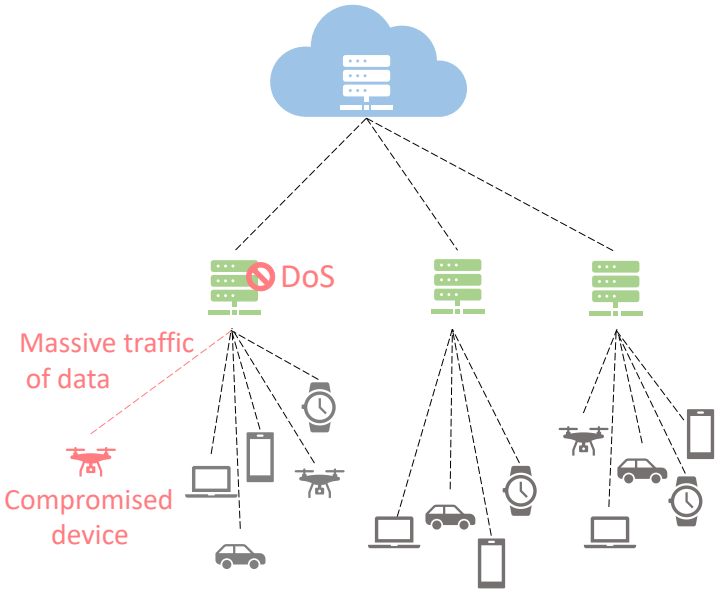

Fig. 2.1 Denial of Service (DoS) attack

1) Exhausting Training Resources During Training: In the training phase of distributed DL systems, exhausting training resources occurs when attackers flood edge or cloud servers with massive traffic, consuming computational power, memory, and bandwidth. This disrupts communication between nodes, delays model updates, and can halt training altogether, preventing the model from being properly trained.

2) Blocking Inference Requests During Inference: During inference, blocking inference requests involves overwhelming servers with traffic, making them unresponsive or slow to process legitimate prediction requests. This poses critical risks in real-time applications, such as autonomous vehicles or healthcare, where delays can lead to system failures[61].

Attack Points for DoS Attacks: DoS attacks can target various points in a distributed DL system [57]:

1) Clients: In distributed DL systems, clients often play a critical role in data collection or local processing (especially in privacy-centric models). Overloading client devices with DoS traffic can disrupt their ability to contribute to the training or inference process, causing delays and reducing system performance. This can also prevent clients from transmitting data to the edge or cloud, further stalling operations.

2) Edge Servers: These servers handle both training and inference tasks closer to the data source. Overloading edge servers can disrupt real-time processing and delay model updates.

3) Cloud Servers: As the central hub for aggregating training data or finalising inference results, cloud servers are prime targets. Attacking cloud servers can halt the entire training process or block critical inference decisions.

4) Communication Links: Attackers can target the network infrastructure connecting clients, edge, and cloud servers, choking data flow and causing system-wide disruptions. This is especially damaging during aggregation in training or when inference results need to be shared across nodes.

Botnet attacks

A botnet is a network of compromised devices, controlled remotely by an attacker, which can be used to perform coordinated malicious activities. These devices are typically infected with malware that allows the attacker to command them to execute tasks such as sending massive amounts of traffic, manipulating data, or performing other malicious operations[62].

In Distributed DL systems, particularly those involving numerous client devices, like in federated learning or IoT-based setups, botnets are relatively easier to deploy due to the decentralized nature of these systems. The widespread use of vulnerable IoT devices, smartphones, and edge devices provides attackers with ample opportunity to compromise these clients and turn them into bots. Once compromised, these devices can act in unison to attack the edge or cloud servers that form the backbone of Distributed DL systems, causing significant disruptions. The decentralized architecture, involving multiple nodes spread across client, edge, and cloud layers, creates many potential attack surfaces, making it easier for attackers to infiltrate the system and scale their botnet[63].

Attacker's Knowledge:

Botnet Attacker's Knowledge: A botnet attacker's knowledge encompasses their understanding of system vulnerabilities, malware design, and network exploitation techniques. The attacker is skilled in identifying insecure devices, such as IoT systems, and deploying malware to compromise them. They know how to propagate infections across devices, creating a large, distributed network of bots capable of receiving and executing remote commands. Additionally, the attacker understands the operational behavior of the targeted system, such as communication protocols, traffic patterns, and resource limitations. This knowledge enables them to craft attacks, such as flooding a communication link or sending malicious data, that exploit specific weaknesses. In federated learning, the attacker uses their understanding of model training and aggregation to manipulate local updates or introduce poisoned data, undermining the integrity of the global model.

Attacker's Goal: The primary goal of a botnet attack in a distributed DL environment is to disrupt the availability of services by overwhelming edge or cloud servers with traffic or malicious activities. The attacker aims to cause delays, system outages, or degraded performance, especially during critical stages like model training or inference. In the context of federated learning, a botnet could also be used to manipulate training data at the client level to induce failures in the global model or sabotage the entire learning process, rendering the system ineffective[64].

Attacker's Ability: An attacker's ability to carry out a botnet attack depends on their control over a network of compromised devices. These devices, called "bots," are typically infected with malware, which allows the attacker to command them remotely. The botnet can be scaled to include hundreds, thousands, or even millions of compromised devices. With these bots under control, the attacker can direct massive amounts of traffic to target servers (e.g., edge or cloud) or manipulate local data at the client level in federated learning environments. This distributed nature of the attack makes it difficult to mitigate, as the traffic originates from numerous legitimate devices[65].

Attacker's Strategies:

Attackers, after forming the Botnet (figure 2.2), can implement different strategies:

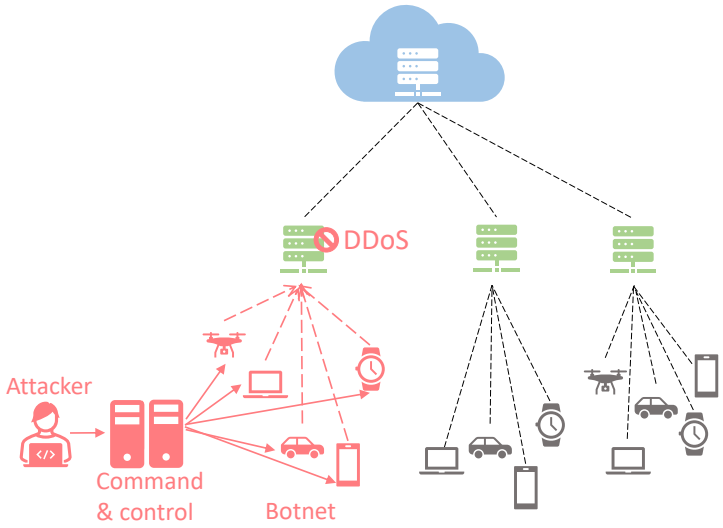

Fig. 2.2 Botnet attack

1) Resource Exhaustion: Botnets can be used to exhaust computational resources by forcing servers to process excessive amounts of data or requests. In the context of DL systems, this might involve sending numerous inference requests or large volumes of fake training data to overload the system, delaying training or causing it to fail[66].

2) Training Data Manipulation: In federated learning, attackers can leverage their control over compromised clients to inject poisoned data into the training process. These compromised devices can send malicious or manipulated training data to the aggregation server, leading to a corrupted global model or hindering the learning process[67].

3) Command-and-Control Attacks: Botnets operate through command-and-control mechanisms, where the attacker issues instructions to all compromised devices simultaneously[63]. In DL systems, the attacker could issue commands to generate coordinated bursts of traffic at specific intervals, disrupting the training or inference process. In federated learning, such attacks can be timed to coincide with aggregation rounds, increasing the likelihood of system failure.

2.1.2 Integrity Threat Model

Integrity threats in Distributed DL refer to attacks that compromise the accuracy and reliability of data, models, or inference results, ultimately leading to the development of unreliable or malicious DL systems[68, 69].

Data poisoning

Data poisoning is an integrity attack where malicious data is injected into the training dataset, causing the model to learn incorrect patterns and make faulty predictions. This attack is particularly harmful in distributed training environments because data is often collected from multiple sources, many of which may be insecure or compromised[70].

Attacker's Knowledge: the attacker's knowledge of the training features and labels is pivotal to the success of the attack. If the attacker has detailed knowledge of the features and labels used during training—such as their distributions, significance, and the relationships between them—they can craft poisoned data that is more likely to be integrated into the training process without detection. For instance, by understanding which features are most influential in the model's decision-making, an attacker can selectively manipulate these features or their corresponding labels to subtly corrupt the model's learning. This knowledge allows the attacker to create poisoned data that either skews the model's predictions (e.g., through label flipping or feature manipulation) or introduces vulnerabilities such as backdoors. In a scenario where the attacker lacks detailed knowledge of the features and labels, the attack might be less precise but could still cause significant damage by overwhelming the system with random or semi-targeted noise, relying on the volume and frequency of the poisoned data to achieve their objectives. The depth of the attacker's understanding of the

training features and labels thus significantly influences the sophistication and impact of the data poisoning attack[71, 70].

Attacker's Goal: The primary goal of a data poisoning attack is to manipulate the model's behaviour during training, either by degrading its overall performance or by influencing its responses to specific inputs. In untargeted attacks, the attacker seeks to broadly reduce the model's accuracy and reliability across a wide range of inputs, leading to poor generalisation and incorrect predictions. This type of attack can render the model ineffective or unreliable for its intended use, compromising the integrity of the system[72]. In targeted attacks, the attacker has a more specific objective in mind. They aim to introduce poisoned data that causes the model to misclassify particular inputs, such as critical data points that are valuable to the attacker. This could involve causing a facial recognition system to misidentify certain individuals or making a fraud detection model overlook specific fraudulent activities. In both scenarios, the attacker's goal is to disrupt the normal functioning of the model in ways that align with their malicious intent, while ideally evading detection[50].

Attacker's Ability The attacker's ability to execute a data poisoning attack depends heavily on their level of access to the training pipeline. In distributed systems, such as those using federated learning or edge computing, the attacker has more entry points. They can compromise specific nodes, like IoT devices, edge servers, or client devices, to manipulate the local training data before it is sent for aggregation [70]. This is particularly concerning in privacy-centric distributed systems, where data remains on client devices and is not centrally monitored, making it easier for attackers to introduce poisoned data at the client side. Regardless of the architecture, the attacker's ability to influence the training process relies on their capacity to corrupt the data being used by the model, either by manipulating it directly at the source or by intercepting it as it travels through the system [73].

Attacker's Strategies: In both data-centric and privacy-centric distributed DL systems, attackers utilise various strategies to corrupt the data before it enters the training pipeline as illustrated in figure 2.3:

1) Label Manipulation (Untargeted Data Poisoning): In this approach, the attacker alters the labels of the training data to cause the model to learn incorrect associations across a broad range of inputs. The goal is not to target specific outcomes but to degrade the overall performance of the model. By mislabeling various data points in the training dataset, the model is trained on false associations, leading to incorrect predictions during deployment[74].

2) Backdoor Attack (Targeted Data Poisoning): In targeted data poisoning, the attacker strategically poisons specific parts of the training data to influence the model's behaviour in particular scenarios, often through the insertion of backdoors. A backdoor attack involves injecting specially crafted patterns into the data that trigger malicious behaviour only when

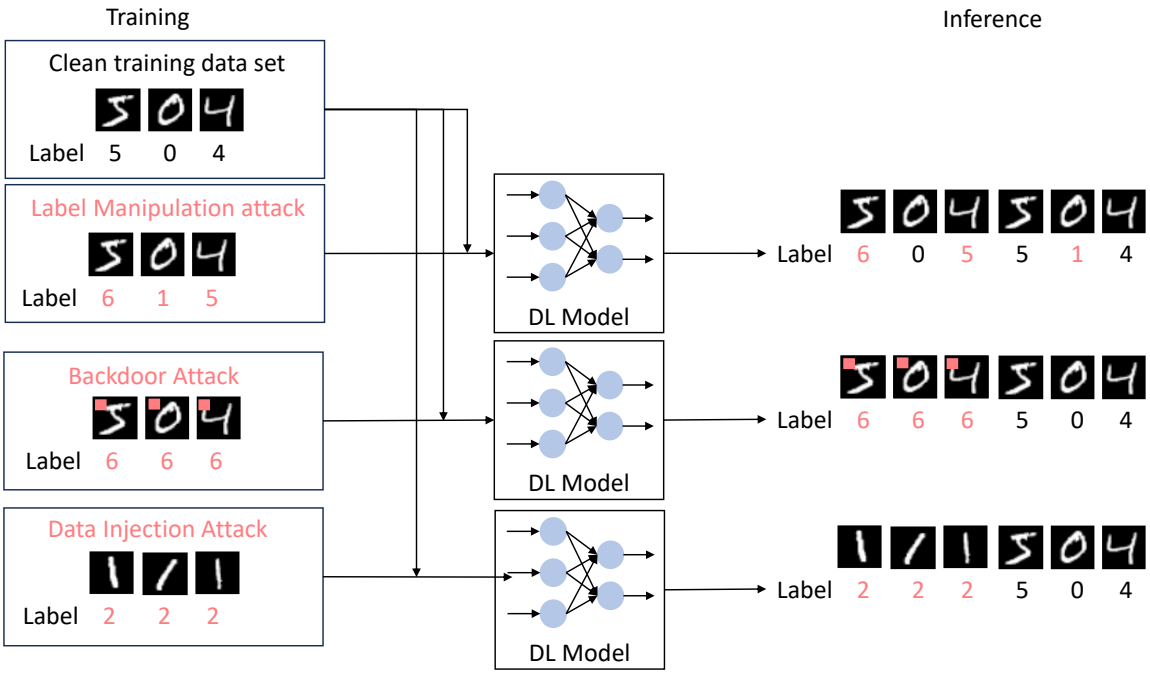

Fig. 2.3 Data Poisoning Attack Strategies

the backdoor trigger is present. For example, an attacker may embed a specific pattern or label combination into the data that will cause the model to misclassify a specific input during inference. This allows the model to perform well on most inputs but misbehave when the backdoor trigger is activated[75].

3) Data Injection: In this strategy, the attacker introduces entirely new, corrupted data into the dataset. This can involve the addition of malicious examples that distort the learning process, causing the model to learn incorrect or harmful patterns. By injecting poisoned data that looks benign but skews the training results, the attacker can degrade the performance of the model without significantly altering the existing training data[76].

Attack entry Points: Entry Points of Data poisoning attack can be at client-Side or server-side.

1) Client-Side Attacks: In both data-centric and privacy-centric distributed DL, the client side represents a vulnerable entry point for data poisoning attacks. In data-centric systems, the attacker corrupts the data before it is transmitted to the edge or cloud for centralised processing[77]. In privacy-centric systems, the attacker targets the local data on the client device, ensuring that poisoned data is used in training without ever being shared beyond the client[73].

2) Server-Side Attacks: In data-centric distributed DL, the edge or cloud servers are also vulnerable to data poisoning attacks, especially during data aggregation or transmission. The

attacker may intercept data at these points and inject poisoned examples into the datasets used for training[77]. In privacy-centric systems, the edge and cloud nodes primarily handle model updates rather than raw data, so these points are less susceptible to direct data poisoning [73].

Model poisoning

Model poisoning is an integrity attack in which an adversary intentionally manipulates the parameters or weights of a deep learning model during the training process. Unlike data poisoning, which corrupts the training data to indirectly influence the model, model poisoning directly alters the internal structure of the model by injecting malicious updates. This results in a compromised model that either misbehaves in specific ways (e.g., misclassifies certain inputs) or subtly embeds a backdoor that can be exploited at a later time.[71]. It is important to note that data poisoning attacks can indirectly lead to model poisoning, as corrupted training data can affect the model's learned parameters, resulting in a compromised model that behaves in unintended ways. Thus, while data poisoning manipulates the input data, model poisoning targets the learning process itself by corrupting the model's internal state[48].

Attacker's Knowledge: The attacker's knowledge of the system plays a critical role in determining the success of model poisoning attacks. In white-box scenarios, the attacker has comprehensive access to the model's architecture, parameters, and training protocols, enabling them to craft highly targeted and effective malicious updates. Conversely, in black-box scenarios, the attacker lacks direct access to the model but may still have some knowledge of the system's general structure and behaviour, allowing them to inject poisoned updates through indirect means, such as data poisoining [78, 79].

Attacker's Goal: The attacker's primary goal in model poisoning is to manipulate the model's behavior by altering its parameters during training. This can involve degrading the overall performance of the model by injecting noise or bias into its learning process. Through these manipulations, the attacker aims to reduce the model's accuracy or reliability, causing it to make incorrect predictions or perform poorly in real-world applications. The subtle changes introduced by the attacker can lead to a compromised model that fails to function as expected under normal conditions[80].

Attacker's Ability: The attacker's ability to influence the training process is determined by their access to the system's infrastructure. In data-centric distributed DL, attackers can compromise nodes at the edge or cloud level, where training occurs in a distributed manner. By gaining control of one or more nodes, the attacker can inject poisoned updates that will be aggregated with updates from other nodes, corrupting the global model[78]. In privacy-centric distributed DL, the attacker can poison local models on compromised clients, which are then aggregated into the global model during the federated learning process. In both

architectures, the attacker's ability to manipulate the training process hinges on their access to the nodes responsible for generating or aggregating model updates.[50].

Attacker's Strategies:

Attackers can employ various strategies to execute model poisoning attacks, each aimed at compromising the integrity of the model during the training process by directly manipulating gradients or model parameters (figure 2.4):

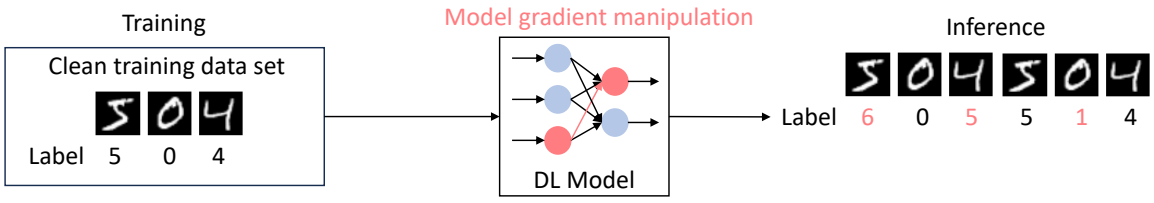

Fig. 2.4 Model Poisoning Attack

1) Sign-flipping attacks: The attacker manipulates the gradients or model parameters by flipping their signs before they are aggregated. By reversing the gradient directions, the attacker can disrupt the learning process and push the model towards an incorrect or malicious state. These attacks are often subtle and difficult to detect, as they can bypass simple aggregation defenses, especially in distributed training environments [23].

2) Gradient noise attacks: The attacker introduces noise into the gradient updates during training. By adding small perturbations to the gradient values, the attacker can degrade the model's performance or steer it toward a malicious state over time. This approach is particularly effective when the noise is introduced gradually, making it hard to detect amidst legitimate gradient updates from other nodes[50].

Attack Points: Model poisoning attacks can occur at several points in the training pipeline, depending on the architecture:

1) Client-side attacks (Privacy-Centric Distributed DL): In systems such as federated learning, model training takes place on potentially insecure client devices. Attackers can poison the local models on these clients before the updates are sent to the edge or cloud for aggregation. The decentralised nature of the system makes it difficult to detect poisoned updates, especially when they come from multiple clients.

2) Edge or cloud-side attacks (Data-Centric Distributed DL and Privacy-Centric Distributed DL): In both architectures, model poisoning can occur at the edge or cloud nodes where the model updates are aggregated. Attackers who compromise these nodes can inject poisoned updates directly into the aggregation process, ensuring that the global model is corrupted. Additionally, the attacker can manipulate the aggregation mechanism itself,

especially if they have knowledge of the aggregation methods used, to amplify the effect of the poisoned updates.

Adversarial samples

Adversarial samples are carefully crafted inputs designed by attackers to deceive deep learning models. These inputs appear normal to human observers but contain subtle perturbations that cause the model to make incorrect predictions. The perturbations are often small enough that they don't affect the input's visible characteristics, but they exploit the model's inherent weaknesses or vulnerabilities in its learned decision boundaries. Adversarial samples are a significant integrity threat, especially in distributed inference, where predictions are made across multiple nodes in a network, making it challenging to detect and prevent such attacks. [81, 82].

Attacker's Knowledge: The attacker's knowledge significantly influences the effectiveness of adversarial sample attacks, typically falling into two scenarios: White-box attacks occur when the attacker has full access to the target model, including its architecture, parameters, and gradients. This comprehensive knowledge enables the attacker to craft highly effective adversarial samples by exploiting the model's specific vulnerabilities. In contrast, Black-box attacks take place when the attacker has no direct access to the model's inner workings but can observe its inputs and outputs. By repeatedly querying the model, the attacker infers its behaviour and creates adversarial samples that deceive the model, even without knowing its internal details[83].

Attacker's Goal: The attacker's primary goal is to manipulate the model into making incorrect predictions by feeding it adversarial samples. This can range from causing the model to misclassify harmless inputs (e.g., mislabeling a stop sign in autonomous driving) to tricking the model into making harmful decisions (e.g., bypassing security systems). The attack can be untargeted, where the goal is to induce any form of incorrect prediction, or targeted, where the attacker aims to manipulate the model into making a specific misclassification (e.g., classifying a malicious input as benign)[84].

Attacker's Ability: The attacker's ability to influence the system depends on their access to the input data and the model's inference process. If the attacker can control or manipulate the input data, such as in edge-based inference systems, they can inject adversarial samples into the model's decision-making process. In distributed inference, adversarial samples can be introduced at any point in the client, edge, or cloud layers, increasing the number of potential attack vectors.

Attacker's Strategies: Several strategies are commonly employed by attackers to generate and deploy adversarial samples (figure 2.5):

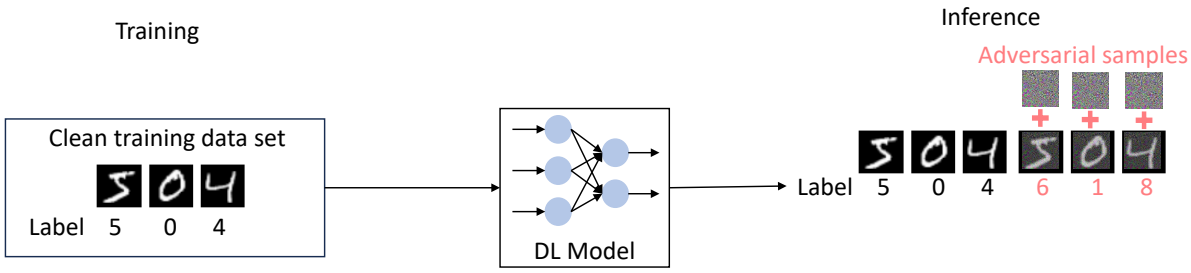

Fig. 2.5 Adversarial Samples Attack

1) Gradient-based attacks: In white-box settings, attackers use the model's gradients to craft perturbations that are imperceptible to humans but cause the model to misclassify inputs. Examples include Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD). These techniques iteratively adjust the input data along the model's gradient to maximise the chance of misclassification[85].

2) Query-based attacks: In black-box scenarios, attackers rely on the model's output to iteratively refine adversarial samples. Techniques like Zeroth-Order Optimization (ZOO) and Square Attack generate adversarial samples without needing access to the model's gradients by probing the model's outputs and inferring patterns from them[86].

3) Patch-based attacks: Attackers create physical or digital adversarial patches that, when placed in the input space, cause the model to misclassify the entire input. For example, a small adversarial patch on a stop sign could cause an autonomous vehicle to fail to recognise it correctly[87].

Attack Points: The attack points for adversarial sample attacks are determined by the architecture of the distributed inference system:

1) Client-side attacks: In privacy-centric distributed DL, adversarial samples can be introduced at the client level, where inference is performed locally. Since the model runs directly on the client's device, attackers can feed adversarial inputs to deceive the local model.

2) Edge-side attacks: In data-centric distributed DL, inference may occur at the edge. An attacker could inject adversarial samples at this level, exploiting the model's weaknesses as it processes the inputs. Given the distributed nature of the system, edge-based inference may have limited capacity to verify the integrity of the inputs.

3) Cloud-side attacks: If inference is offloaded to the cloud, attackers can still introduce adversarial samples at earlier stages in the inference pipeline (e.g., client or edge levels) or during data transmission to the cloud. Cloud-side attacks exploit the model's global

decision-making process, often with a more significant impact due to the aggregation of inputs from multiple sources.

2.2 Defense Mechanisms Against Security Threats in Distributed DL

This section presents various defense mechanisms tailored to counter attacks that threat availability and integrity of Distributed DL systems.

2.2.1 Availability Defense Mechanisms

DoS and Botnet Detection

DoS and botnet detection are critical for maintaining continuous service availability in distributed DL systems. These methods typically involve the deployment of monitoring tools and anomaly detection systems to identify and mitigate the impacts of Denial of Service (DoS) attacks and botnet activities. Botnets, which consist of compromised devices acting under the control of a central attacker, can be used to launch large-scale DoS attacks by overwhelming servers with excessive traffic, rendering distributed DL systems unresponsive [88]. By rapidly detecting these threats and initiating appropriate countermeasures, these systems play a crucial role in preserving system functionality during malicious attacks.

Research in this area is ongoing, with many studies proposing distributed DL-based detection techniques specifically tailored for securing distributed DL environments. For privacy-centric distributed DL systems, federated learning-based detection methods are particularly well-suited, as they allow for decentralised threat detection while maintaining user privacy[89]. Conversely, in data-centric distributed DL systems, parallelism-based detection methods are more appropriate, as they capitalize on the distributed nature of training and processing across edge and cloud nodes, enhancing detection accuracy and efficiency. These advanced detection methods are vital in ensuring robust protection against evolving threats in distributed DL infrastructures.

In the context of federated learning, Zhang et al.[32] propose a DoS attack detection model called FLDoS, which is based on federated learning. This model addresses the challenges of data privacy and imbalance in distributed environments. The attack detection is implemented using a combination of a hierarchical aggregation algorithm based on K-Means and a data resampling technique. These methods help mitigate the global and local data imbalances that can hinder DoS detection performance. FLDoS improves detection accuracy

by allowing clients to train models locally on their own data while aggregating the results centrally without sharing sensitive information. The attack detection is specifically designed to combat DoS attacks, enhancing detection performance in scenarios where data distribution is non-IID (non-Independent and Identically Distributed).

Similarly, Doriguzzi et al.[33] propose FLAD (Adaptive Federated Learning Approach to DoS Attack Detection), a federated learning framework designed to detect Denial of Service (DoS) attacks in cybersecurity while addressing challenges such as non-IID (non-Independent and Identically Distributed) and unbalanced data. The attack detection is implemented by leveraging an adaptive mechanism that dynamically assigns computational workloads to clients based on the performance of the global model on their local validation data. This strategy allows the system to prioritize clients with harder-to-learn attack profiles, speeding up convergence and improving detection accuracy without sharing sensitive data. The framework specifically targets DoS attacks, enabling robust and efficient detection across various types of DoS traffic.

Botnet detection has also been successfully implemented in the federated learning context. popoola et al.[34] introduce a federated learning approach for detecting zero-day botnet attacks in IoT-edge devices. This method preserves data privacy by allowing IoT devices to train models locally and then aggregate the models using the FedAvg algorithm without sharing sensitive data. Attack detection is implemented using a deep neural network (DNN) architecture designed for network traffic classification. The model is trained across multiple IoT-edge devices, detecting various zero-day botnet attacks, such as DoS, reconnaissance, and data theft, through collaborative learning without compromising privacy. This approach is particularly effective against zero-day botnet attacks, ensuring high detection rates and minimising false positives while maintaining data security and privacy.

For data-centric distributed deep learning, researchers have introduced various approaches to detect DoS and botnet attacks. Reddy et al.[35] proposes a hybrid neural network architecture for the early detection of DoS attacks using deep learning models. The attack detection is implemented through a combination of two deep learning approaches: a Gradient Boosting Decision Tree (GBDT) model to classify spatial features of network traffic and a lightweight convolutional neural network (CNN) model to capture temporal features. The outputs of these models are merged using an ensemble learning method to classify network traffic as either benign or malicious. This approach effectively detects Denial of Service (DoS) attacks, ensuring early detection by analyzing both spatial and temporal traffic patterns, ultimately improving detection accuracy and reducing false positives.

A study by Vasiliadis et al. [36] introduces MIDeA, a multi-parallel intrusion detection architecture designed for high-performance detection of network attacks in high-speed

environments. This system achieves scalability and enhanced performance by distributing the workload across multiple processing units, including multi-queue NICs, multi-core CPUs, and GPUs. Attack detection is implemented using stateful packet analysis and pattern matching for identifying signatures of various network intrusions. MIDeA focuses on signature-based detection, which excels at identifying known attacks with low false-positive rates. The architecture is particularly effective in detecting a wide range of network attacks, including Denial of Service (DoS) and other high-volume threats, by performing efficient traffic inspection at rates exceeding 70 Gbit/s using commodity hardware.

To the best of our knowledge, there is limited research that leverages parallelism in distributed DL for detecting DoS and botnet attacks. Given that botnet formation often leads to DoS attacks, there is a critical need for distributed, accurate, and rapid DL-based detection methods capable of identifying botnets in their early stages of formation. DL-based botnet detection in distributed deep learning systems should utilise the same parallel mechanisms inherent to these systems, ensuring efficient and effective detection that aligns with the distributed architecture

2.2.2 Integrity Defense Mechanisms

Defenses against integrity attacks on DL models are generally divided into two main categories: data-driven and model-driven approaches [90]. Data-driven defenses focus on processing or detecting poisoned data before it is introduced to the model. For example, You et al.[91] propose an adversarial defense technique known as De3 (Denoising by Decoder for Adversarial Defense), which mitigates adversarial effects by first adding Gaussian noise to adversarial samples and then reconstructing the original images. Similarly, Zhu et al.[92] introduce a method to detect adversarial data by generating denoised versions of the original samples for classification. If the classifications of the denoised and original images differ, the input is flagged as adversarial. Another study by Shi et al.[93] proposes the removal of backdoor triggers through image transformation, followed by using a pre-trained diffusion generative model to restore the image's semantic integrity. These data-driven defenses typically function independently of the underlying learning architecture [90].

In contrast, model-driven defenses focus on constructing or modifying models to make them inherently more resilient to adversarial attacks. This section will specifically review model-driven defense methods, as the primary focus of this thesis is on enhancing the robustness of DL models during training and inference within distributed DL environment

Adversarial Training

Adversarial Training (AT) is a common defense method against adversarial examples. Adversarial Training involves training the model with data that includes carefully crafted adversarial inputs designed to trick the model into making incorrect predictions[94]. Adversarial Training has been extensively researched in the context of centralised deep learning[95]; however, its application in distributed learning is still in its early stages.

Zhang et al. [37] propose a Distributed Adversarial Training (DAT) framework that enables scalable, large-batch adversarial training across multiple machines to improve the robustness of deep neural networks (DNNs) against adversarial attacks. By leveraging distributed systems, the framework allows for large-batch training, where batch sizes are scaled proportionally to the number of computing nodes, addressing the challenge of maintaining robust accuracy and efficiency in large-scale training. To manage the complexities of large-batch training, DAT incorporates a layer-wise adaptive learning rate (LALR) to stabilise the optimisation process, allowing for higher learning rates without degrading performance. Additionally, each distributed worker in the DAT framework independently generates adversarial examples using attack generation methods such as Projected Gradient Descent (PGD) or Fast Gradient Sign Method (FGSM), computes the gradients locally, and transmits the gradients to a central server for aggregation and model updates, ensuring both scalability and robustness in adversarial settings.

Federated Adversarial Learning extends this concept to federated learning environments. Li et al. [38] propose a framework for Federated Adversarial Learning (FAL), aiming to enhance the robustness of deep neural networks against adversarial attacks in a federated learning setting. In FAL, adversarial samples are generated locally on client devices, which are then used for adversarial training through a min-max optimisation process. A central server then aggregates locally updated models by using the FedAvg algorithm. The FAL framework addresses several unique challenges in federated learning, such as low convergence rates, inter-client heterogeneity, and the security concerns of adversarial attacks. Through the use of techniques like gradient coupling, the paper provides a convergence analysis for the proposed framework in over-parameterised neural networks, showing that with appropriate learning rates and communication rounds, the minimal training loss can converge to a small value even in the presence of adversarial perturbations.

Building on the idea of enhancing federated learning, Chen et al. [39] propose a framework for Federated Adversarial Learning (FAL) that integrates randomised smoothing to enhance the certifiable robustness of models trained in a federated setting. During the Attack Generation and Gradient Computation process, adversarial examples are created locally on client devices using a smoothed version of the classifier, which perturbs inputs with

Gaussian noise. The gradients are then computed using either a stochastic gradient estimator, which involves back-propagation on multiple corrupted samples, or a one-point gradient-free estimator, which is computationally lighter but has a higher variance.

Similarly focused on federated learning, Shah et al. [40] proposes an adversarial training framework tailored for federated learning settings where communication constraints and non-IID (non-identical and independently distributed) data are significant challenges. The Attack Generation and Gradient Computation process in this framework involves each client generating adversarial samples locally using Projected Gradient Descent (PGD). These adversarial examples are optimised through Stochastic Gradient Descent (SGD) to maximise the loss on perturbed inputs. Each client then locally updates its model using these adversarial examples, ensuring that the resulting models are robust against attacks. After local updates, the model parameters are aggregated at a central server using either FedAvg or FedCurv fusion algorithms to synchronise the models across clients. This distributed setup aims to solve the minimax problem, enhancing the robustness of the global model across different clients, despite the communication constraints and data heterogeneity.

To the best of our knowledge, Adversarial Training has not yet been explored within the framework of hierarchical federated learning.

Model Discrepancy-based Robust Aggregation Methods

Robust aggregation methods are developed to securely aggregate model updates in the federated learning framework, mitigating the effects of data and model poisoning during the training process [51]. Algorithms are utilised to prevent the global model from being compromised by outliers or malicious updates from compromised nodes. To address malicious updates caused by data and model poisoning, a range of robust aggregation algorithms has been introduced, which identify model discrepancies between benign and malicious models through similarity-based aggregation techniques [96]. Many robust aggregation methods have been developed for traditional federated learning [51], but few studies have focused on their application in hierarchical federated learning.

Zhao et al. [41] aim to address poisoning attacks in Collaborative Edge-Cloud Federated Learning (CEFL) systems by proposing a novel defense mechanism named FlexibleFL. The study examines poisoning attacks and their defense strategies in practical CEFL scenarios, highlighting the necessity for flexible and efficient outlier detection techniques. FlexibleFL calculates participant contributions to assess model quality and assigns edge models accordingly. This method indirectly penalises potential attackers, establishing an effective defense mechanism suitable for cloud-edge environments. Zhao et al. [41] employ the cosine similarity metric as part of defense mechanism to evaluate the variability between

the weighted features of different participants. This evaluation helps in determining each participant's performance, constraining subsequent model training rounds to enhance the global model's quality.

In a related study, Zhou et al.[23] propose a robust framework called RoHFL for hierarchical federated learning in Internet of Vehicles (IoV) systems, designed to defend against poisoning attacks from malicious vehicles and compromised Road Side Units (RSUs). The study focuses on mitigating poisoning attacks originating from malicious vehicles and RSUs, emphasising robustness in the federated learning framework. RoHFL includes a reputation-based robust model aggregation scheme, logarithm-based normalisation for gradient updates, and a reputation score updating mechanism. These features enhance the system's defense capabilities, offering improved performance compared to prior methods. RoHFL uses a similarity-based scheme for updating reputation scores, applying the cosine similarity metric to compare the aggregated gradient with each local logarithm-normalized gradient, effectively identifying and mitigating the influence of malicious nodes.

Similarly, Zhou et al.[97] focuses on designing a hierarchical Federated Learning (FL) framework named Robust and Privacy-Preserving FL (RoPPFL) for edge computing, ensuring both privacy preservation and robustness against poisoning attacks. The study examines poisoning attacks and offers a significant contribution by combining privacy preservation with a similarity-based robust model update aggregation scheme, which maintains model utility even under attack. The research utilises the similarity metric to evaluate the trustworthiness of client updates, comparing the similarity between the poisoned model and the original model. This similarity value is used as an aggregation weight to counteract the negative effects of poisoning attacks.

Another related study by Zhou et al. [42] proposes a robust hierarchical federated learning (R-HFL) framework to bolster system resilience against abnormal behaviours while optimising communication efficiency in cloud-edge-end cooperative networks. This framework operates within a hierarchical structure and employs a detection mechanism called Partial Cosine Similarity (PCS) to filter out malicious clients by assessing the similarity between previously established edge models and newly uploaded client models. The study addresses model and data poisoning attack scenarios. It evaluates the effectiveness of R-HFL under different PCS thresholds, successfully detecting and mitigating malicious models to ensure that only trustworthy clients contribute to the training process.

Although these studies provide robust aggregation methods for hierarchical federated learning, there is still a need to improve their effectiveness. The model discrepancy detection primarily relies on a single metric, cosine similarity. To enhance these methods, further

analysis of their efficiency in hierarchical federated learning is necessary, along with the exploration of additional model discrepancy metrics

Model pruning and cleaning

Model pruning is a technique used to clean the model from redundant or less significant parameters from a model [98]. By simplifying the model and reducing its complexity, pruning can enhance robustness against various attacks, such as model poisoning, data poisoning, and especially backdoor attacks [43]. Typically applied post-training, before deployment, model pruning ensures that the final model is both efficient and secure.

In centralised DL, Wang et al.[43] propose a system for detecting and mitigating backdoor attacks in deep neural networks (DNNs). It reverse-engineers triggers used by backdoors and mitigates them through model pruning, which removes neurons responsible for the backdoor's behaviour. The approach is validated across various attack scenarios, including those that inject backdoors through poisoned training data or direct modifications to a pre-trained model, effectively disabling backdoors while maintaining model performance and robustness against adversarial data.

Similarly, Liu et al. [44] propose a defense mechanism against backdoor attacks in deep neural networks by combining model pruning and fine-tuning into a technique called fine-pruning. This method first prunes neurons that are dormant when processing clean inputs, thereby removing backdoor neurons that only activate with malicious inputs. After pruning, fine-tuning is applied to restore the model's performance on clean data. Fine-pruning effectively disables backdoors introduced during training by removing the capacity for backdoor triggers while maintaining high accuracy on benign inputs. This approach specifically targets backdoor attacks, where malicious triggers are embedded in models during outsourced training.

On the other hand, model pruning has been successfully applied to mitigate backdoor attacks in distributed deep learning, particularly within the federated learning framework. Meng et al. [45] proposes FLAP (Federated Learning with Data-Agnostic Pruning), a post-aggregation model pruning technique designed to enhance the Byzantine robustness of federated learning (FL). This approach focuses on improving security against malicious attacks in FL, such as model poisoning, without relying on client cooperation or training data. Model pruning is implemented after the aggregation step, where insignificant and dormant parameters, which are often introduced by poisoning attacks, are pruned to disable their harmful effects. FLAP targets Byzantine attacks and model poisoning by reducing the impact of adversarial inputs while preserving the fidelity of the global model. The technique is particularly effective against adversarial models in federated learning settings.

In a similar vein, research by Wu et al.[46] centers on federated learning and proposes a method to mitigate backdoor attacks in federated learning using a combination of federated pruning and extreme weight adjustments. The approach starts with federated pruning, which removes redundant and "backdoor neurons"—neurons that trigger malicious behaviours upon recognising backdoor patterns—while keeping neurons that respond to clean inputs intact. After pruning, fine-tuning is applied to recover any accuracy lost on benign data due to the pruning process. Finally, the extreme weights of the neurons are adjusted, particularly in the last convolutional layers, to further mitigate backdoor attacks by limiting abnormal neuron activation. This approach is especially effective against backdoor attacks introduced through poisoned data in a federated learning system, without requiring access to clients' private data.

Despite the successful application of model pruning in traditional federated learning, there remains a research gap in studying the efficiency of model pruning within distributed deep learning mechanisms, including hierarchical federated learning.

2.3 Security Assessment of Distributed DL

Beside the defence mechanisms, the security assessment of distributed deep learning systems is essential for ensuring their robustness in adversarial environments. This assessment can be divided into two categories [99]: first, evaluating the system's resilience under adversarial attacks and defensive mechanisms. This evaluation focuses on how well the system can preserve its accuracy and functionality when confronted with various threats. Second, examining the system's dynamics during attacks, which provides insights into how adversarial actions impact the learning process. This dual approach allows for a comprehensive evaluation of distributed deep learning systems by not only measuring their performance in challenging scenarios but also uncovering the underlying vulnerabilities that adversaries may exploit, ultimately guiding the development of more robust and trustworthy system[99].

The study by Usama et al.[47] explores the vulnerabilities introduced by machine learning (ML) models, particularly in the context of 5G and beyond networks. It emphasises the impact of adversarial machine learning attacks on the reliability and robustness of ML-driven solutions in key areas like resource allocation and network automation. The research highlights the importance of evaluating performance under attack by demonstrating how adversarial attacks compromise the functionality of supervised, unsupervised, and reinforcement learning models. Additionally, it underscores the need to understand the dynamics of the training process under adversarial influence, offering valuable guidelines for enhancing the robustness of ML models in distributed environments.

Similarly, Shejwalkar et al.[48] focuse in their research on performance under attack by evaluating the impact of poisoning attacks in real-world federated learning environments. It specifically looks at untargeted poisoning attacks, where adversaries corrupt model updates to degrade overall system performance. The study shows that simple, low-cost defenses such as norm-bounding can substantially mitigate these attacks, even in large-scale production settings. Moreover, it contributes to understanding training process dynamics by examining real-world conditions and challenging theoretical assumptions, thus providing a more practical framework for defense strategies in federated learning systems.

Ibitoye et al.[49] contribute to performance under attack by testing the resilience of deep learning-based intrusion detection systems (IDS) in IoT networks. It compares the effectiveness of two neural networks—FNN and SNN—under adversarial perturbations, showing that normalisation techniques like self-normalisation can improve robustness. The study also offers insights into the training process dynamics, particularly how different model architectures respond to adversarial influences, advancing the understanding of IDS robustness in distributed IoT settings.

Bhagoji et al.[50] also addresse performance under attack, particularly in federated learning environments. It investigates targeted model poisoning attacks, where a single malicious agent manipulates the model's training to cause misclassifications. The research highlights the dynamics of the training process by exploring strategies such as boosting and alternating minimisation, demonstrating how adversaries can maintain stealth and effectiveness even when defenses like Byzantine-resilient aggregation are in place. This study underscores the need for more sophisticated defense mechanisms that account for the evolving nature of attacks throughout the training process.

Finally, Li et al.[51] examines performance under attack by evaluating the robustness of several Byzantine-resilient aggregation schemes against various attack strategies like noise injection and label flipping. It also delves into the training process dynamics by analyzing how Non-IID data scenarios exacerbate performance drops, despite the presence of robust aggregation techniques. The introduction of ClippedClustering, which mitigates the effects of amplified malicious updates, further enhances understanding of how training dynamics can be optimised to withstand adversarial attacks.

To the best of our knowledge, there are no studies that exclusively focus on analysing the dynamics of distributed deep learning systems under attack. Most research tends to emphasise performance evaluation and defense mechanisms, rather than the system's dynamic behaviour during the training process. However, studies that incorporate dynamic analysis offer valuable insights into the evolving nature of attacks and their impact on system performance over time.

2.4 Summary and Positioning of this Dissertation

This dissertation investigates critical security challenges in Distributed DL systems across the client-edge-cloud continuum, focusing on both distributed training and distributed inference processes. The research addresses the unique security threats in two key categories of Distributed DL: Data-Centric Distributed DL (Chapter 3) and Privacy-Centric Distributed DL (Chapter 4 and 5). Through three interconnected projects, this work explores methods to safeguard the integrity and availability of DL models against various adversarial attacks, with each chapter contributing to different aspects of these challenges.

In Chapter 3, titled *Modular Neural Network for Edge-based Detection of Early-stage IoT Botnet*, the focus is on ensuring availability when clients (e.g., IoT devices) are sending their data for analysis, inference, or model training at the edge in a Data-Centric Distributed DL system. This chapter introduces EDIT, an edge-based modular neural network (MNN) system designed to detect early-stage IoT botnets before any attack is launched. The system leverages multi-access edge computing (MEC) to analyse network traffic at the edge of the network, identifying anomalies indicative of botnet activity. Operating within the client-edge-cloud continuum, EDIT is highly efficient, detecting botnets with a lower false-negative rate and achieving detection times as fast as 16 milliseconds. This chapter also emphasises the use of parallelism techniques (MMN) to suit the distributed nature and real-time requirements of these systems, allowing for fast and accurate detection. It highlights how edge-based Distributed DL can prevent botnet-induced Denial of Service (DoS) attacks in real-time, playing a critical role in maintaining service availability.

Chapter 4, titled *Security Assessment of Hierarchical Federated Deep Learning*, shifts the focus to Privacy-Centric Distributed DL, addressing integrity concerns during the training process of Hierarchical Federated Learning (HFL). The research evaluates the security of HFL systems under adversarial attack scenarios, specifically investigating their resilience to both inference-time and training-time attacks. The hierarchical structure of HFL provides robustness against untargeted training-time attacks, but vulnerabilities emerge in the face of targeted backdoor attacks, particularly when malicious clients exploit the overlap in coverage areas between edge servers. This chapter emphasises the dual nature of HFL's security—while it offers strong defenses using adversarial training, it remains susceptible to targeted integrity attacks, underscoring the need for improved security strategies within HFL systems.

Continuing the investigation of HFL security, Chapter 5, titled *Dynamics of Hierarchical Federated Learning Under Attacks*, further explores integrity concerns, focusing on the dynamics of HFL systems under a range of complex attack scenarios. The research introduces a novel Model Discrepancy score, which combines multiple metrics—Dissimilarity,

Distance, Uncorrelation, and Divergence—to quantify discrepancies in model updates and detect malicious activity. Through a comparative analysis of 3-level and 4-level HFL architectures, the study demonstrates that the 4-level HFL architecture encounters challenges in maintaining server clustering quality, as the overlapping discrepancy patterns between benign and malicious servers make differentiation more difficult, primarily due to the added aggregation level. This chapter emphasises the importance of comprehensive discrepancy management and robust aggregation techniques to enhance the security and robustness of HFL systems under attack.

Taken together, these three projects address the core security challenges of availability and integrity in Distributed DL systems across the client-edge-cloud continuum. Chapter 3 focuses on maintaining system availability during inference at the edge in data-centric DL environments by detecting and mitigating botnet attacks in their early stages. Chapters 4 and 5 concentrate on the integrity of hierarchical federated learning during training in privacy-centric DL environments, evaluating the resilience of HFL systems to adversarial attacks and providing insights into the model discrepancy phenomenon during training under adversarial attacks. Figure 2.6 presents a taxonomy of threats and defenses in Distributed DL across the client-edge-cloud continuum, clearly positioning this dissertation's contributions within the framework. Specifically, it illustrates how this work addresses critical security concerns, including availability during inference at the edge (Chapter 3) and integrity during federated learning training (Chapters 4 and 5).

This dissertation is positioned within the broader domain of Distributed DL across the client-edge-cloud continuum, with a specific focus on enhancing the security of DL systems in federated learning and edge computing. By offering novel approaches such as EDIT for botnet detection and the Model Discrepancy score for enhancing robustness in HFL, this research contributes to the development of more resilient and secure Distributed DL systems.

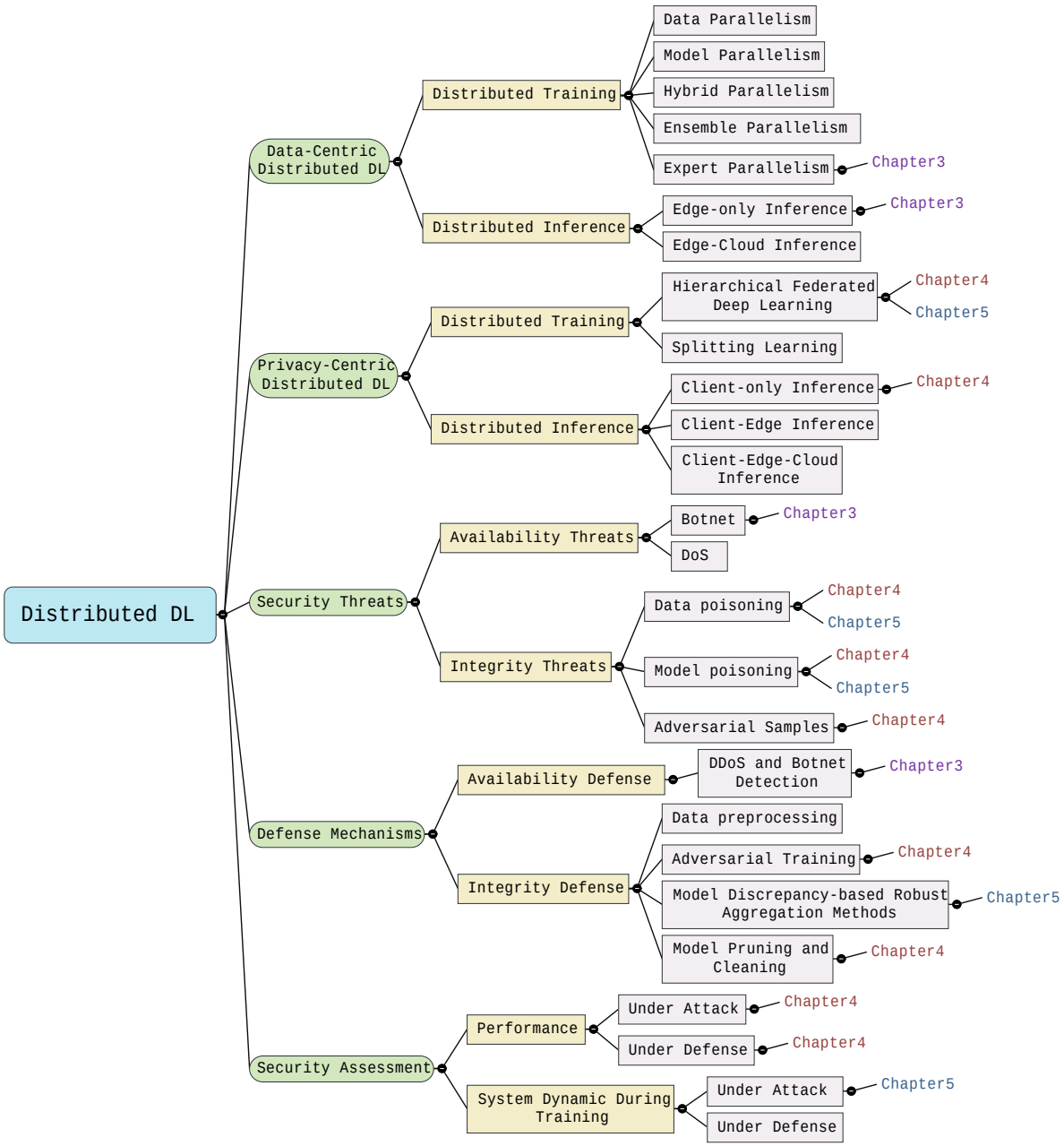

Fig. 2.6 The taxonomy of threats and defenses in Distributed DL across the client-edge-cloud continuum and the position of this dissertation's contributions

Chapter 3

Modular neural network for Edge-based Detection of early-stage IoT Botnet

The Internet of Things (IoT) has led to rapid growth in smart cities. However, IoT botnet-based attacks against smart city systems are becoming more prevalent. Detection methods for IoT botnet-based attacks have been the subject of extensive research, but the identification of early-stage behaviour of the IoT botnet prior to any attack remains a largely unexplored area that could prevent any attack before it is launched. Few studies have addressed the early stages of IoT botnet detection using monolithic deep learning algorithms that could require more time for training and detection. We, however, propose an edge-based deep learning system for the detection of the early stages of IoT botnets in smart cities. The proposed system, which we call EDIT (Edge-based Detection of early-stage IoT Botnet), aims to detect abnormalities in network communication traffic caused by early-stage IoT botnets based on the modular neural network (MNN) method at multi-access edge computing (MEC) servers. MNN can improve detection accuracy and efficiency by leveraging parallel computing on MEC. According to the findings, EDIT has a lower false-negative rate compared to a monolithic approach and other studies. At the MEC server, EDIT takes as little as 16 milliseconds for the detection of an IoT botnet.

3.1 Context

Smart cities integrated with sensors are the peak of urban innovation in the digital era. They are important for improving the quality of public services and people's lives. A smart city is dependent on the Internet of Things (IoT) and edge/cloud computing for operation. The term "Internet of Things" refers to the ability of devices to talk to each other over the

Internet and other networks using sensors built into the devices. Each IoT device in smart cities keeps track of how it is being used and what is going on around it and shares this information with its network. This includes information gathered from residents, buildings, public transportation vehicles, and healthcare facilities, among other domains. The huge quantities of data collected by IoT devices in smart cities are sent to servers in either an edge or cloud environment, where they are analysed to improve the management of facilities, equipment, and services. As smart cities proliferate, the attack surface expands due to the expansion of billions of IoT devices connected to each other and to edge and cloud servers. Attackers might exploit IoT device vulnerabilities to form an IoT botnet and launch attacks against servers in the edge or cloud environment, with potentially catastrophic consequences for the smart city's systems.

IoT botnet-based attacks are continually becoming a serious concern in smart cities. IoT botnets are widespread; they spread quickly and have the ability to do more damage than any other kind of disruptive behaviour. This is primarily due to the fact that an IoT botnet is a collection of compromised IoT devices that a bot master (attacker) can use to execute commands through a simple process of IoT botnet formation and attack execution [100, 101]. One example of such an IoT botnet was in 2016, when a botnet known as Mirai malware was responsible for a Distributed Denial of Services (DDoS) cyber-attack on the servers of Dyn. (Dyn is a company that administers the majority of the Internet's Domain Name System (DNS) infrastructure [88].) This attack brought down a large number of websites in Europe and the United States, notably Twitter, Reddit, and CNN. In this incident, about 100,000 IoT devices were compromised.

Besides Mirai, different malware families that exclusively target IoT devices have been released, e.g., Bashlite, Tsunami, Hide and Seek, BrickerBot, Luabot, and Hajime. Each IoT botnet family has distinctive behaviour that represents the family's characteristics and features. However, generally, IoT botnet malware families carry out their attack tasks in three main phases. First, the botnet is formed (through scanning and spreading), then it is commanded and controlled (CnC), and finally, the attack is carried out. The first and second phases are considered the early stages of the IoT botnet before launching the attack, whereas the last stage is the late stage of the IoT botnet. In the botnet formation process, the bot master scans vulnerable IoT devices and infects them with bot malware, which then propagates and spreads across the network to make other IoT devices infected with bot malware. Then, the commands given to a botnet to control it by attackers are either to retrieve information from a bot or to conduct the intended attack (e.g., DDoS). Lastly, the intended attack is conducted toward the target victim in order to damage the confidentiality, integrity, or availability of the network [102, 103], leading to a failure that affects some of

the smart city nodes with catastrophic consequences and alters the normal behaviour of smart cities. The main difference between the second and the last stage is that the second stage focuses on preparation and coordination, while the last stage involves executing the planned malicious activities. The successes in IoT botnet-based attacks emphasise the importance of implementing robust IoT botnet detection methods.

Smart city security has received a lot of attention in recent years for protecting smart cities from IoT botnet-based attacks [104]. Machine learning (ML) and deep learning (DL) have been the dominant choices in the development of intrusion detection systems (IDS) for the monitoring and detection of IoT botnets by analysing network communication patterns [100]. Several studies are available on ML-based IoT botnet detection that aims to detect late-stage IoT botnets (e.g., DDoS) [105, 106]. They detect the abnormalities in the network communication patterns of IoT devices after attacks have already been launched. However, the early stages need more attention so that the IoT botnet can be detected before launching any IoT botnet-based attack. Only a few studies addressed the detection of early stages of IoT botnets (discussed in Section 3.2) using ML and DL approaches to differentiate between several early stages of IoT botnets as a multi-classification problem. For instance, Hegde et al. [107] examined botnet detection via experiments with supervised ML and DL classifiers. Alzahrani et al. [108] aimed to develop a multi-classification Neural Network model using FastGRNN to detect the IoT botnet in a short time. Abdalgawad et al. [109] analyse network traffic to identify IoT botnet behaviour using Adversarial Auto-encoders (AAE) and Bidirectional Generative Adversarial Networks (BiGAN). Wazzan et al. [110] apply Cross CNN LSTM to identify the propagation of IoT botnets.

These ML and DL-based IoT botnet detection studies, despite achieving high accuracy, have several shortcomings:

- Although these studies rely on detecting the early-stage IoT botnet from network communication patterns, they lack in the realm of failure detection and recovery, where early-stage IoT botnets should be modelled as severe fault conditions that can be observed in the network communication of smart city systems.

- These studies do not examine the idea that the early stages (Scan and CnC) could occur simultaneously. It is possible that the reported effects on network communication of smart city systems have more than one origin (Scan and CnC). In order to take the most appropriate action, it is crucial to distinguish between these two stages.

- Since these studies are based on deep machine learning models for multi-class classification of IoT botnet stages, these models are large in size and demand a lot of resources

to train. Therefore, the models take a long time to construct and update, which could make smart cities vulnerable to a zero-day attack.

- A significant delay occurs in detecting the early stages of IoT botnets. Consequently, the attacker may have the ability to launch an attack on any smart city node. Normal behaviour in smart cities will be interrupted as a result.

- The architecture that these studies suggest has a high rate of false negatives; as a result the attacks are missed and remain undetected, which makes IoT-based smart cities vulnerable and not well protected.

To address the mentioned shortcomings, this chapter answers the following two sub-questions under Research Question 1 of this thesis:(**Research Questions1.1**) How early stages of the IoT botnet can be modelled as faults in network communication in smart city systems? (**Research Questions1.2**) How can a DL model be trained to differentiate between the characteristics of these faults in a short time to reduce false negative rates without significant detection delay?

To answer these research questions, we investigate how to distinguish the two early stages of the IoT botnet (Scan and CnC) as faults in smart city network communication that affect the network's normal behaviour (benign communication) by examining network traffic characteristics travelling during normal function and IoT botnet behaviour in smart city systems using a supervised Neural Network (NN). We first modelled the early stages of the IoT botnet as fault conditions in the network communication of smart city systems (Research Questions1.1) and then used the fault characteristics to train a DL model for IoT botnet detection.

To ensure effective detection and prevention of Distributed Denial of Service (DDoS) attacks, the scale of the botnet becomes a critical factor. For smaller systems, compromising 10-15% of devices might be sufficient, while larger-scale systems often require over 30% of devices to be compromised. Moreover, the general timeline for a botnet attack spans hours to days, as the early stages of scanning, spreading, and CnC setup can take hours, and the attack itself can be launched almost instantaneously once the botnet reaches the desired size. Thus, addressing these threats requires a proactive approach that operates in real-time to identify and mitigate threats before they escalate.

We propose Modular Neural Networks (MNN) as a DL approach for early-stage IoT botnet detection that integrates modularity into the neural network design. Unlike ensemble methods such as cascade or boosting, which rely on sequential refinement where models depend on each other, MNNs operate with independent specialised modules that work in parallel, making them faster and more suitable for real-time detection. Cascade and boosting

methods focus on improving accuracy by iteratively correcting errors, often requiring more computational time, while MNNs prioritise efficiency and modularity by allowing individual modules to process sub-tasks independently, significantly reducing overhead and enhancing scalability for edge environments. This approach simplifies the detection of IoT botnet stages by treating each stage as a separate and specialised detection module. The modular design enables parallel training and detection, allowing updates—such as refining models or incorporating emerging threat patterns—to be processed on-the-fly without disrupting ongoing operations. By leveraging Multi-Access Edge Computing (MEC) servers, these modules allow for rapid detection and prevention at the edge, where IoT devices are directly connected, avoiding delays caused by centralised processing. Fig. 3.1 illustrates the differences between training monolithic NN and MNN for multi-classification detection problems.

This design ensures real-time adaptability to evolving threats, continuous protection, and operational continuity. It addresses the shortcomings of traditional approaches by reducing false negatives, enhancing detection speed, and preventing system vulnerabilities caused by delays or static updates. Fig. 3.2 demonstrates the deployment of MEC servers in smart cities, where IoT devices are connected to MEC servers, enabling fast and efficient processing.

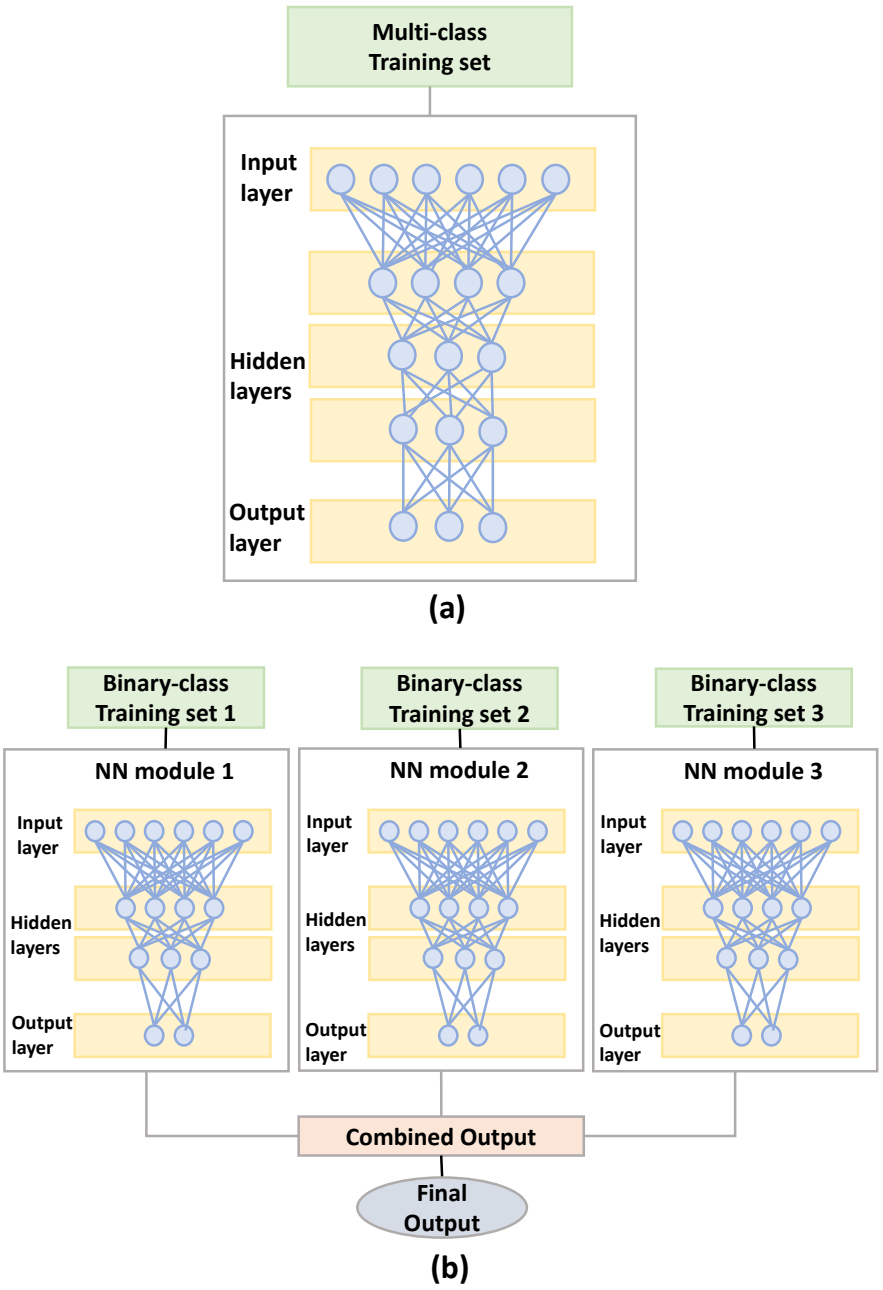

Fig. 3.1 (a) Monolithic NN for multi-classification detection problem (b) MNN for multi-classification detection problem

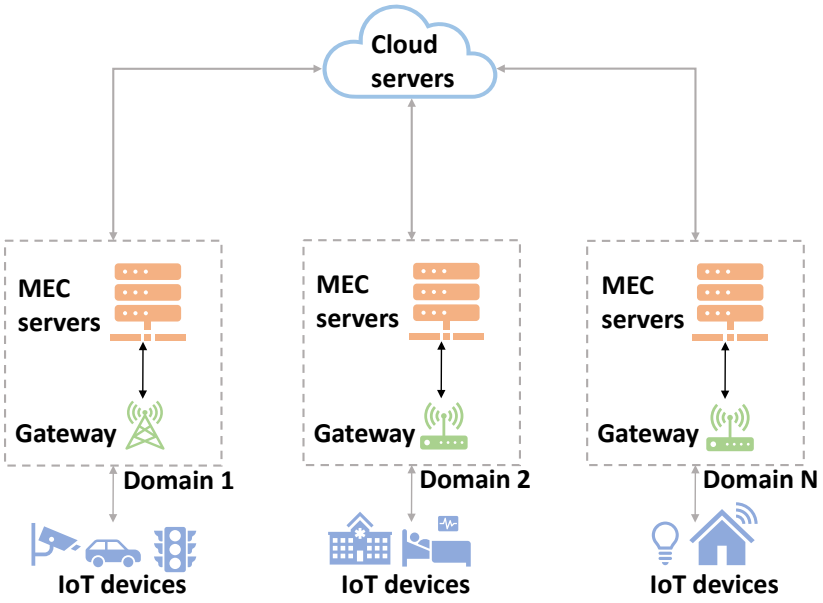

Fig. 3.2 Multi-access edge servers in smart cities

In summary, our contributions to address Research Question 1 and its two sub-questions in this thesis are as follows:

- We model the early stages of the IoT botnet (Scan and/or CnC) as faults that cause abnormalities in network communication traffic in smart cities.
- We propose an **e**dge-based **d**etection system for early-stage **I**oT botne**t** (EDIT) in smart cities using modular neural network. In this chapter, we provide the first effort at using a modular neural network (MNN) to identify communication traffic faults caused by IoT botnets in their early stages. For fault recognition (Scan and/or CnC), a quick training and detection process, and accurate detection, the MNN technique breaks the detection task into separate parallel binary classification tasks that are run in parallel on the MEC server.
- Through a series of experiments on publicly accessible datasets, we compared our MNN-based approach to many state-of-the-art methods, demonstrating the extent to which the detection rate was improved while still maintaining a fast training and detection speed. We found that the MNN approach had a false-negative rate of zero.

The remainder of the chapter is organised as follows: Section 3.2 introduces the related work. In Section 3.3, early-stage IoT botnets (Scan and CnC) are modelled, and the

characteristics of network communication during these stages are described. Section 3.4 presents the proposed EDIT system and describes its components that depend on the MNN approach. The evaluation methodology and the experimental setting are described in Section 3.5. In section 3.6, the evaluation of EDIT results is discussed in terms of effectiveness and efficiency. Section 3.8 concludes that our EDIT system is able to take advantage of the parallelism of the modular approach to improve both the speed and accuracy of IoT botnet detection.

3.2 Related Works

In Section 2.4.1, we highlight research on DoS and botnet detection in Distributed DL systems, with a focus on ensuring system availability in a broader context. However, in this section, we specifically discuss in detail the related work that aligns with the objectives of this study presented in this chapter.

A few researchers used a range of ML/DL approaches to detect IoT botnets at an early stage. In their efforts to increase the performance of IoT botnet detection using ML/DL algorithms, researchers have developed datasets or used publicly available datasets. There is also some research which applies similar principles of MNN in terms of task decomposition to detect the early and late stages of IoT botnets using a multi-model-based ML/DL approach. In this section, we will first discuss the early-stage IoT botnet detection research and then the research that applies the multi-model ML approach to train and detect IoT botnets.

3.2.1 Early-stage IoT botnet detection

In [111], the authors obtained a collection of labelled IoT behavioural data, which comprised both legitimate IoT network traffic and botnet malicious traffic, using an IoT medium-sized network design (83 IoT devices). To gather data, three well-known botnet malware (Mirai, BashLite and Torii) infections and connections with CnC nodes were employed. The dataset collected, called MedBIoT, is used later by researchers. A binary ML classification model is applied to divide the acquired data into two separate categories (e.g., benign and malware). The authors used Random Forest (RF), K-Nearest Neighbours (K-NN), Decision Tree (DT), as well as Support Vector Machine (SVM) algorithms. RF, K-NN, DT, and DT models showed an accuracy of 0.9532 (for RF), 0.9025 (for KNN), and 0.9315 (for DT). The result of the SVM model was not mentioned because it showed much less accuracy than other models.

The authors of [112] conducted different experiments. The relevant experiment to our work is the classification of malicious packets as a binary task for botnet detection. The authors investigated and analysed three recurrent deep-learning methods.: FastGRNN, LSTM, and GRU. The results of the experiment have an area under curve score of 99.9% for MedBIoT and 99.75% for the Kitsune dataset.

Using openly available datasets (IoT23), the research in [107] examined botnet detection via experiments with supervised ML and DL classifiers. Using this dataset, the researchers were able to examine the effectiveness of the classifiers on a dataset that included both benign and malicious data. The accuracy rate was 85.2%, according to the data. As shown in this study, adding more benign network traffic packets to the IoT training dataset increases the performance of classifiers by up to 99.9%, as shown in this study.

The study [108] aimed to develop a multi-class classification NN model that could be trained fast, identify changes in real-time and be accurate. FastGRNN beat the other models in the studies on the MedBIoT and RGU datasets by lowering training and detection time while keeping a high F1 score. Using FastGRNN, the whole test set was identified in 29 seconds. Furthermore, for MedBIoT multi-classification, the model achieved competitive F1 scores of 99.99% and 99.04% for RGU multi-classification.

In [113], distinct ML and DL methods were tested using various datasets (N-BaIoT, IoT-23, Kitsune, MedBIoT). Among these models were Logistic Regression, K-Nearest Neighbors, naive Bayes Decision Trees, and Random Forests. The researchers discovered that the random forest technique was the most accurate and fastest of all four datasets evaluated. An identical experiment was performed using MLPN and LSTM. In terms of accuracy, LSTM surpasses Multi-Layer Perception Network (MLPN) with 99.9%, 99.8, 91.1% for N-BaIoT, IoT-23, and Kitsune datasets, respectively. However, MLPN required significantly less training and detection time.

The study in [109] demonstrates that network data analysis may be utilised to identify malicious behaviour when using generative DL algorithms like Adversarial Auto-encoders (AAE) and Bidirectional Generative Adversarial Networks (BiGAN). The IoT-23 dataset was used to construct generative models to detect distinct threats connected to early-stage IoT botnets. As the research shows, generative models won out over traditional ML approaches. Both the AAE and the BiGAN-based models attained F1-scores of 99%.

An IoT botnet detection method is presented in the research [110]. The botnet's early stages were studied, and a baseline ML model was developed for detection. Using a combination of DL models for a Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM), the researchers developed an effective detection technique called Cross

CNN LSTM to identify the IoT botnet. Studies have shown that this model is 99.7% accurate, outperforming other ML techniques using the MedBIoT dataset.

In study [114], one-class KNN was utilised to detect malicious IoT botnets. The main aim was to address the computing power in IoT devices by developing a lightweight IoT botnet detector using ML. The performance of ML classifiers was enhanced by employing filter and wrapper techniques to choose the appropriate features. Using the filter and wrapper techniques, various datasets were able to reduce feature space. According to the findings, the authors' proposed method was effective in detecting an IoT botnet with an F1-score ranging from 98 to 99% for various IoT botnet datasets.

3.2.2 Multi-model ML for IoT botnet detection

In [115], the proposed system for IoT botnet detection was divided into subsystems. Each subsystem is responsible for detecting one kind of IoT botnet malicious activity sequentially. A classifier is trained to differentiate between malicious and benign behaviour for each kind. A classifier decides if the data input is malicious or not. If it is malicious, the system raises an alarm. Otherwise, the output is passed to the next classifier. The new classifier takes the output of the previous classifier if the output is not a malicious activity, and makes a decision again. This continues until one of the classifiers raises an alarm or until the last classifier decides that the data input is benign. An artificial neural network as the classifier is trained and used for each kind of specific attack. According to these study results, all of the classifier accuracy is up to 99%.

The authors in [115] continue their work of [116] where they apply the feature selection approach, which makes the system less computationally expensive. They also deployed the same system with a "hybrid" classification, in which each system is trained with a different algorithm. The results show that hybrid classification can also achieve high accuracy.

In the study [117], the authors proposed a two-fold ML method to detect IoT botnets. The detection task was decomposed into two parts. The first step is scanning vulnerable devices for activity detection. In this step, a model was trained to classify normal and scan behaviour. The second step is for detecting DDoS activity, and another model was trained using another dataset of normal and DDoS activity. The results show that the proposed method can achieve an overall accuracy of 98%. However, the study only shows the overall accuracy and does not show how the final detection decision of the two classifiers could be combined.

The study [118] proposed a collaborative ML model for early-stage botnet activity detection. The three heterogeneous feature sets are collected from network flows, system logs, and system resources. The detection task was decomposed between three ML models. Therefore, a separate ML model was trained for each input feature set to differentiate

malicious and benign behaviour during the early stages of the IoT botnet life cycle. Then, all the results from the 3 classifiers are integrated to make the final decision by a majority of votes. The result shows that the proposed method gives roughly 99% accuracy.

The author of [119] introduced the EDIMA, a home network-focused IoT botnet detection tool. It uses a two-step process to identify bots communicating with an edge gateway: first, it detects scanning activity in the gateway's overall traffic using machine learning; and then it uses bot-CnC communication traffic patterns to identify specific bots using Autocorrelation Function (ACF). The components of EDIMA are as follows: a parser for network traffic, an extractor for features, a machine learning-based bot detection, a policy engine, a model function for machine learning, and a PCAP database for malware. Performance evaluation results using the testbed configuration with real-world IoT malware traffic and other public IoT datasets show that EDIMA has very low false positive rates for bot scanning and bot-CnC traffic detection.

The quantities of computation that can be traded off vary depending on the multi-task technique applied in these studies. In [115],[116], [117], [119], sequential multi-task models are implemented. As a result, they are not readily capable of being parallelised to decrease the duration of training. Conversely, the multi-task machine learning (ML) model suggested by [118] can be executed in parallel. Nevertheless, it incurs substantial resource consumption, including high memory and CPU utilisation, as it simultaneously gathers three distinct feature sets that have characteristics different from each other.

On the other hand, despite the fact that the multi-task model method was used in the preceding studies, they do not reflect MNN-based detection system designs in the real sense of modularity, in which the system design consists of a number of specialised modules. These modules should display the following properties [120]:

- The modules have specialised computational architectures to recognise and respond to subsets of the total detection task.
- Each module differentiates one network behaviour from another, independently of the other modules.
- Modules have a simpler architecture than the system as a whole.
- The basic module outputs are combined to achieve the complex overall system response.

In contrast to prior research, our MNN architecture breaks down detection into numerous binary-classification tasks, each of which can be handled by a dedicated sub-neural network.

On the MEC server, the sub-neural networks are set up in parallel, and their predictions are combined together to achieve the task with high performance and low resource consumption. This approach was chosen over other architectures, such as monolithic models or sequential ensemble methods like boosting and cascades, due to its ability to operate efficiently in real-time environments. MNNs excel in modularity, utilising independent, lightweight, and specialised modules to process tasks simultaneously, significantly reducing detection latency, computational overhead, and increasing accuracy.

3.3 Early-Stage IoT Botnet Model

IoT botnets' principal targets are IoT devices. This form of botnet takes advantage of the fact that IoT devices generally have limited system resources and skimp on security. In general, botnets are conducted in three phases. First, the botnet formation; second, commanding the botnet; and third, conducting the attack. During the botnet formation, the bot master scans vulnerable IoT devices and infects them with bot malware. The bot is then listed on the command and control (CnC) server, and the bot malware spreads through the network, infecting other IoT devices with bot malware to make a botnet. In the second phase, the commands given to a botnet by attackers are mainly of two types: message commands and malicious commands. The message command is to retrieve information from the bot. Whereas the malicious commands are issued to a bot to act maliciously toward the target victim (any network node) and conduct the intended attack. These two phases are the early stages of IoT botnet tasks. In the third phase, the botnet performs the intended attack (e.g., DDoS) in order to damage the confidentiality, integrity, or availability of the network [102]

The three phases mentioned earlier are typical for IoT botnets, and they may be found in many different families of IoT botnets. Here, we'll highlight the features and characteristics of one of the IoT botnet families, Mirai, that is being analysed in this study.

Mirai is a malware family that infects IoT devices before launching DDoS attack. As shown in Fig. 3.3, we provide a model of its fundamentals as well as how it propagates.

According to authors in [121], the Mirai botnet performs seven tasks (also shown in Fig. 3.3):

1. **Scan.** The initial bot used in an attack will use Telnet's TCP ports 23 and 2323 to send TCP SYN scanners to random IPv4 addresses. The bot will try to log into a vulnerable IoT device through Telnet with a variety of user names and passwords if it finds one.

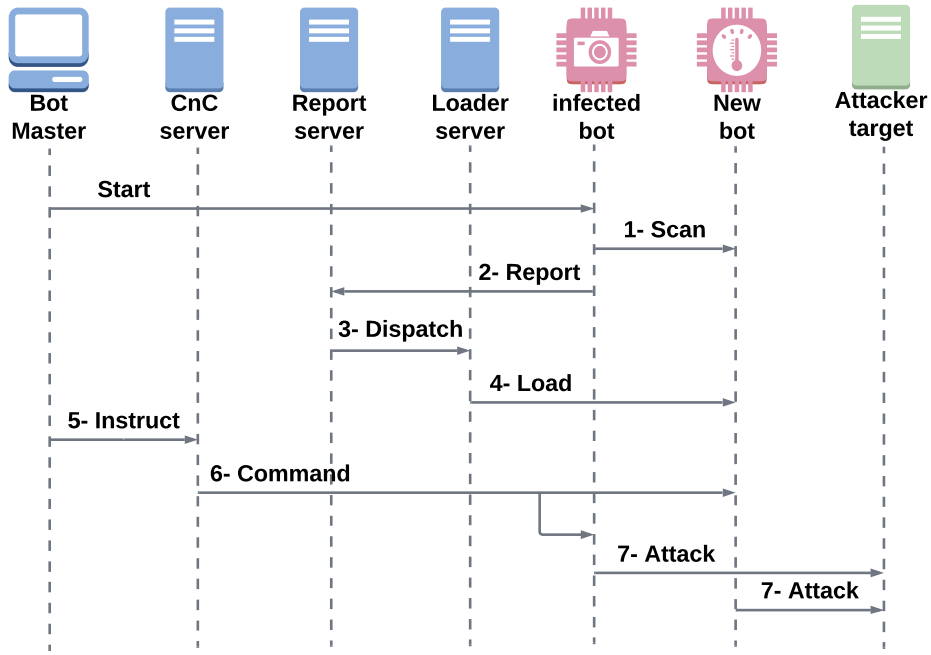

Fig. 3.3 Mirai tasks

2. **Report.** After the bot has shown its authenticity for the first time; it will communicate the IP address of the vulnerable IoT device together with the credentials it was given to a report server.

3. **Dispatch.** IP addresses of the vulnerable IoT devices, together with the credentials, are dispatched to the loader server. The load server builds an application to log into the devices in order to determine the sort of operating system and CPU architecture of the device. Then, the loader prepares the malware that can be run on the vulnerable device.

4. **Load.** Infected IoT devices download and run malware sent by the loader server. The malware has been customised for the architecture of the infected device. Infected IoT devices will execute scans to look for additional vulnerable devices and will take commands from a command and control server. Therefore, the botnet is composed of a variety of Internet of Things devices that have been compromised.

5. **Instruct.** Through the CnC server, the attacker, gives instructions to the botnet so that it may carry out the attack as planned.

6. **Command and Control(CnC).** The command to carry out the predetermined attack is relayed from the CnC server to the compromised IoT devices that are part of the botnet.

7. **Attack.** During this phase of the mission, the botnet will initiate an attack against the target.

Early stages of IoT botnet tasks (Scan, CnC) on smart cities often cause unusual network traffic that puts the network communication channel in a faulty state. The faulty state can be persistent and correlated with abrupt changes in traffic-related features. A network packet is a fundamental unit of network traffic that passes from sender to receiver through a network. In addition to the actual data, which is known as the payload, packets also contain the header and the trailer. The choice of a set of network communication features that can be used to find IoT botnets is one of the biggest problems researchers have to deal with. By analysing the network packets that traverse during normal and abnormal (IoT botnet) communication and searching for any anomalous behaviour patterns, we are able to determine the communication behaviour features that contribute to the faulty state. To extract the features of the communication behaviour of IoT botnets, it is necessary to sniff packets originating from the network. While monitoring network traffic, we are focusing exclusively on the packet headers and ignoring the packet's payload and trailer. Packets with the same destination IP, source IP, destination port, source port, and protocol are aggregated as a flow. The header of each packet in a flow can be used to figure out statistics about the packets. Abrupt changes refer to significant changes in the network packet feature statistics.

3.3.1 Characteristics of the network communication faulty state due to early-stage IoT botnet

In this study, detection of the early-stage IoT botnet depends on the characteristics and features of network communication traffic that is sent or received by the infected IoT devices during Mirai botnet execution. The abnormal network traffic from IoT devices is our main emphasis. Accurate analysis of the early stages of a Mirai botnet on IoT devices is feasible by continuously gathering network traffic feature statistics such as protocol type, packet size, packet duration, and other associated flow-based statistics. While performing the scan task and the command task, data is collected from the network and then extract statistical characteristics to use as input to a deep learning algorithm for model training.

In this section, we describe the possible network-based manifestations of the Mirai botnet's scan task and command task:

Scan task

The majority of IoT botnets employ spreading tactics that are strikingly similar to those employed by worms. They will automatically conduct scans and brute-force attacks after infecting a device in an attempt to infect other machines [122]. Scan task conducted by the Mirai botnet has three characteristics: a TCP port scan target, random stateless IP scanning, and weighted dictionary credential access.

Mirai starts by scanning a set of predefined TCP ports. The code fragment associated with malware scan capabilities reveals that 90% of scans target TCP port 23 and 10% target port 2323 [123].

The scanning strategies used by bots are completely random and stateless. The IP addresses of the destination are generated at random and assigned to the scan packets. The same value is also assigned to the field corresponding to the TCP packet sequence number [123, 122]. This strategy is called the random stateless scanning strategy. In this strategy, the botnet will keep scanning even if the vulnerable IoT device does not react. When the bot receives a response from the vulnerable IoT device, it first sends a rest packet and attempts to compromise the device [88].

After identifying a vulnerable IoT device, the Mirai botnet employs a weighted dictionary attack, also known as brute force, to get access to it. In a weighted dictionary attack, the bot is outfitted with a database of commonly used passwords. Random credentials are selected and tested against the vulnerable IoT device [88].

It is possible that the aforementioned scan behaviour features represent anomalous network traffic rather than benign network traffic. The packet measurements may or may not readily reveal these anomalies. Consequently, flow statistical data accurately characterise scan activity more precisely. First, a clear indicator of the abnormal behaviour is the increase in the number of flows entering the network as a result of a large number of unique source IP addresses and scanning ports. Second, to scan as many devices as possible, malware frequently sends a small number of scanning packets to the same IP address. Thirdly, throughout the scanning process, a bot will randomly establish a huge number of TCP connections to IP addresses, the vast majority of which will be ignored and hence remain partially open. Consequently, there will be a large number of TCP half-open connections if we examine their frequency. In addition, the length of malware scanning packets is often shorter than that of normal benign TCP packets. Scan packets feature fewer transmission latency than packets from benign IoT devices.

Command and Control (CnC) task

Once an IoT device has been infected, it transforms into a bot, registers with the CnC server, and joins the botnet. The bot is awaiting commands from the CnC server to control its behaviour. Each botnet family uses its own protocol messages and communication mechanism to carry out this command [122].

The communication mechanism used by attackers consists mainly of two types of commands: message commands to retrieve information from the bot and malicious commands that are issued to a bot to act maliciously toward the target victim (any network component) and conduct the intended attack (e.g., DDoS) [102].

Mirai has a unique protocol based on hexadecimal messages for its communication mechanism. The bot uses a handshake process to communicate with the botnet's command and control server; the first message is a registration packet with the hexadecimal value 0x00,0x00,0x00,0x01. To initiate the heartbeat procedure, the botnet CnC receives a string representing the system architecture after 10 seconds and immediately responds with an acknowledge message by delivering a pulse packet containing the hexadecimal string 0x00 0x00 [122]. The CnC server then begins sending malicious commands to the infected IoT device. The bot will execute the attack coded inside the packet's seventh byte (the payload) after it has been received. In addition, bytes 9–13 represent the Internet Protocol address of the victim server. The attack's duration is encoded in bytes 3-6 [124].

Various sorts of traffic abnormalities can be identified by analysing the aforementioned command behaviour characteristics. Each bot establishes a TCP connection with the CnC server and transmits and receives periodic TCP heartbeat messages. The timing of communication exchanges is periodic, according to [119]. Moreover, when an IoT device communicates with a public server IP address, the server sends packets back to the relevant IoT device. Therefore, it is possible to keep track of how frequently an IoT device sends and receives packets from the public server. The size of the payload in packets travelling between IoT devices and the CnC servers is also statistically observable and can be more or less than the size of the payload in benign traffic.The payloads are presumably encrypted, although certain characteristics can be derived from them to determine the CnC connection behaviour. Therefore, we solely evaluate characteristics extracted from packet headers and completely disregard payloads.

3.3.2 Definition of the network communication faulty state due to early-stage IoT botnet

From the discussion in the previous section, we can define the normal and abnormal behaviour during the early-stage IoT botnet. The main objective of this work is to develop a technique for distinguishing between benign and early-stage IoT botnets (Scan, CnC) in a smart cities domain. From the graph theory, the domain is represented as a graph $G = (V, E)$, where V is the set of nodes and E is the set of directed communication channels. V is made up of m nodes $\{v_1, ...v_m\} \cup g$. These nodes represent IoT devices, and all nodes are connected to these devices in the edge, cloud and the internet through an edge gateway g. Two nodes $v_i, v_j \in V$ exchange messages through communication channels $e_{i,j} \in E$. Each $v_i \in V$ communicates directly to the gateway g through communication channels $e_{i,g}$, where $e_{i,g} \in E, \forall v_i \in v_1, ..., v_m$. The state of each channel $e_{i,j}$ denoted by $S^{e_{i,j}}$, is defined over a set of state features, f_n. Using the snapshot of the state of the channel, we can derive a set of measurements that describe the channel state and performance metrics (e.g., delay) that could contribute to the channel's faulty state. Thus channel $e_{i,j}$ state is given by:

$$S^{e_{i,j}} = \{f_1, f_2 ... f_k)\}.$$

For each state feature f_n of $S^{e_{i,j}}$, we denote $min(f_n)_t$ and $max(f_n)_t$, the expected minimum and maximum limits, respectively, of the value of f_n, at a time unit t. Using these bounds, we can now define the normality and abnormality of the communication channel as follows:

Definition 1 (Normal) A state $S_t^{e_{i,j}}$ of channel $e_{i,j}$ at time t is *normal* $\iff$:

$$\forall f_n, f_n \in [min(f_n)_t, max(f_n)_t].$$

In other words, the state of a channel is normal if all the state features are within the expected limits. We assume the channel between the node and the gateway always exists. On the contrary, in an abnormal state, such conditions may not hold, as explained in Definition 2.

Definition 2 (Abnormal) A state $S_t^{e_{i,j}}$ of channel $e_{i,j}$ at time t is *abnormal* $\iff$:

$$\exists f_n, f_n < min(f_n)_t \vee f_n > max(f_n)_t.$$

By definition, an abnormal state may include features outside the minimum and maximum limits.

Following Definitions 1 and 2, we say that the domain state is normal if $\forall e_{i,j} \in E$ are normal (Benign) due to normal execution $\varepsilon, \forall v_i \in V$. Similarly, we characterised a domain

state as abnormal if $\exists e_{i,j} \in E$ is abnormal as a result of abnormal execution $\varepsilon \in [Scan, CnC]$ from Scan and Command and Control execution of a node.

3.3.3 Problem definition of the network communication faulty state detection

Detection systems can usually be configured to network communication faulty state by monitoring for simple thresholds and limit boundaries. However, each stage in the IoT botnet reveals different actions; thus, the scan and CnC behaviour of the IoT botnet can affect many features of the communication channels. Besides the differentiation between normal (Benign) behaviour and abnormal behaviour (Scan, CnC), discrimination between abnormal behaviours (Scan and CnC) is very critical to take suitable action. Thus, the tracking of these behaviours is not simple, and a probabilistic discrimination function based on expected network communication behaviour and pattern should be deployed.

Problem 1: Given a state $S_t^{e_{i,j}}$ of channel $e_{i,j}$ in an execution ε at time t, a discrimination function f can accurately classifies the state $S_t^{e_{i,j}}$ as Benign, Scan or CnC.

3.3.4 Communication channel state features

A similar real-time smart cities domain testbed has been developed consisting of real and virtual IoT devices. The features of the communication channel state in the domain have been extracted during benign, scan and CnC behaviour of Mirai botnet. In this study, we will utilise these two data sets (IoT23 and MedbIot) that contain IoT botnet statistical network packet features.

Table 3.1 exhibits the feature extracted by [125] and from the header fields of network packets while observing the benign, scan, and CnC communication behaviour. This dataset (called IoT23) consists of 20 features.

Another feature set is collected for IoT botnet behaviour called MedBIoT. According to [111], MedBIoT is a 100-feature dataset. These features are divided into four categories: 1- traffic originating from the same MAC and IP (Host-MAC and IP features), 2- traffic between specific hosts (channel features), 3- traffic between specific hosts, including ports (socket features), and the 4-time interval between packets in channel communication (network jitter features). The number of packets, average packet size, and standard deviation are all computed for each broad classification. In addition to the standard statistical measures of frequency, mean, and standard deviation, the correlation coefficient (PCC) of packet size, radius, covariance, and magnitude for channel and socket categories were also derived. Table 3.2 shows the description of these features.

Table 3.1 Feature description of IoT23 dataset.

	Feature
1	Flow start time
2	Unique ID
3	Source IP address
4	Source port
5	Destination IP address
6	Destination port
7	Transaction protocol
8	Service http, ftp, smtp, ssh, dns, etc.
9	Total duration of flow
10	Number of payload bytes from source
11	Number of payload bytes from destination
12	Connection state
13	connection source locally or remotely
14	connection destination locally or remotely
15	Missing bytes during transaction
16	History of source packets
17	Number of packets from source
18	Number of IP level bytes from source
19	Number of packets from destination
20	Number of IP level bytes from destination

3.4 EDIT: Edge-based Detection of Early-Stages IoT Botnet System

This section describes the general architecture of the proposed EDIT system. The architecture shown in Fig. 3.4 relies on two components: (a) *MNN-based model constructor* and (b) *IoT botnet detector* at the edge.

The training task is the responsibility of the constructor. The training task is subdivided into three binary sub-tasks: 1) Benign and Scan classification, 2) Benign and CnC classification and 3) Scan and CnC classification. The MEC server stores flow-based statistics data from IoT network traffic that contains the features of benign, scan, and CnC behaviour and uses this data to train a model. After the neural network modules are trained, the detector at the MEC server uses the trained modules. The detector is responsible for detecting IoT botnets in their early stages by monitoring the group of IoT devices linked to the same MEC server. This section elaborates on the proposed EDIT system and its components.

Table 3.2 Feature description of MedBIoT dataset.

Categories	Features	Numbers
Host-MAC&IP	Packet count, mean, variance, magnitude,	15
Channel	Packet count, mean, variance, magnitude, radius, covariance, and correlation	15
Network Jitter	Packet count, mean and variance of packet jitter in channel	35
Socket	Packet count, mean, variance, magnitude, radius, covariance and correlation	35
Total		100

3.4.1 MNN-based Model constructor

For the MNN-based model constructor, pre-collected flow-based statistics data from the IoT network traffic that represents the characteristics of benign, scan and CnC behaviour is stored and used as an IoT botnet training dataset. To deploy the modularity on this detection task, the constructor first applies task decomposition (as explained in [126]) and then performs the training job.

Task decomposition

The IoT botnet detection task is divided into sub-tasks (modules) according to IoT botnet behaviours. Our system aims to differentiate between three types of behaviour classes, namely benign, scan, and CnC. Each module is responsible for detecting a class of botnet behaviour against another class:

Lets C_i for $i = 0, 1, \ldots, 2$ be the classes of IoT botnet behaviours (Benign, Scan, and CnC).

$$C = \{C_0, C_1, C_2\}.$$

We call this a 3-Class IoT botnet behaviour detection task. The training dataset for each class DS_0, DS_1 and DS_2 for Benign, Scan, and CnC respectively can be written as:

$$DS_i = \{(X_i, Y_i), \} i = 0, 1, 2.$$

where X_i and Y_i are feature vectors and class labels of the samples specific to the class C_i.

The scope and organisation of publicly accessible datasets vary. It is thus essential to preprocess and aggregate the data into the proper format for ML/DL. Therefore, the min-max

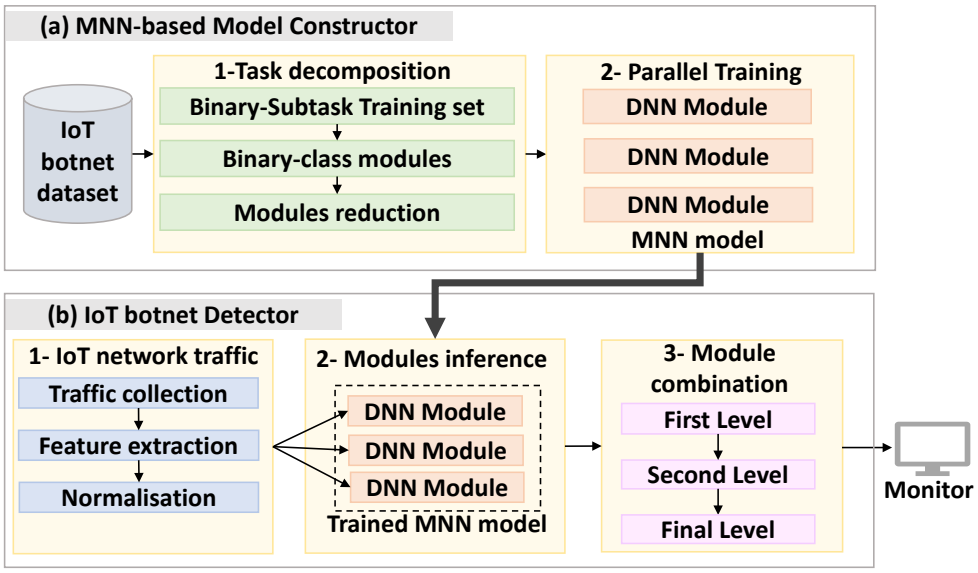

Fig. 3.4 Architecture of EDIT system

approach was used to normalise the features in all sub-datasets DS_0, DS_1 and DS_2 ensuring that no one feature dominates the model training stage. This is accomplished by the use of the following equation:

$$x_{normalised} = \frac{x - min(x)}{max(x) - min(x)}.$$

That is necessary to obtain a normalised dataset of DS_0, DS_1 and DS_2. Therefore, we can write the IoT botnet training dataset T as a union of sub-tasks class training datasets: DS_0, DS_1, and DS_2 as:

$$T = \cup_{i=0}^{2}(X_i, Y_i),$$

For MNN, we decompose the 3-Class task into M binary-class modules. Each binary-class M_{ij} module is trained to classify a class C_i from a class C_j for $\forall\ i$ and $j \in \{0,1,2\}$ and $i \neq j$. Such a decomposition produces $M = 6$ binary-class modules $(M01, M02, M10, M12, M20, M21)$. The training dataset for binary-class module M_{ij} that differentiates a class C_i from a class C_j is

$$T_{ij} = \{(X_i, 1)\} \cup \{(X_j, 0)\},$$

which means all feature vectors of class C_i are labelled as 1, and feature vectors of class C_j are labelled as 0.

In this decomposition, for each binary classification task for classes (C_i) and (C_j), there are redundant modules, i.e., either module M_{ij} or module M_{ji} is sufficient for classifying

class (C_i) from class (C_j). That is, training sets T_{ij} and T_{ji} are complementary to each other:

$$T_{ij} = \{(X_i,1)\} \cup \{(X_j,0)\} \quad \text{complements} \quad T_{ji} = \{(X_j,1)\} \cup \{(X_i,0)\}.$$

In such case, we reduce the number of sub-tasks by merely using an inverse unit (INV) on the output of M_{ij} to derive the output of M_{ji} as:

$$M_{ji} = INV(M_{ij}).$$

This will lead to the reduction of the number of modules to $\binom{K}{2}$.

Therefore, the number of modules is reduced to 3. As modules (M_{10}, M_{20}, M_{21}) can be obtained from modules (M_{01}, M_{02}, M_{12}) using INV units. Ultimately, there are 3 modules (M_{01}, M_{02}, M_{12}) and 3 training datasets (T_{01}, T_{02}, T_{12}) for Benign against Scan, Benign against CnC, and Scan against CnC, respectively.

Table 3.3 shows the modules M_{ij} and the corresponding module training dataset T_{ij}.

Table 3.3 The modules and the corresponding dataset.

Classes	Benign vs Scan	Benign vs CnC	Scan vs CnC
Module	M_{01}	M_{02}	M_{12}
Training Set T_{ij}	**{X_0,1}∪{X_1,0}**	**{X_0,1}∪{X_2,0}**	**{X_1,1}∪{X_2,0}**

DNN Modules Parallel Training

For training the modules (M_{01}, M_{02}, M_{12}), we have applied a simple feed-forward deep neural network (DNN). The architecture of the DNN model included an input layer, densely interconnected hidden layers,dropout layers, and an output layer. The number of neurons in the input layer (n) is proportional to the amount of training data features that accurately characterise a single network traffic packet sample x. The number of densely interconnected hidden layers h and the number of neurons in each of them is often determined by testing several sets of values in order to develop the best DNN architecture that delivers the maximum classification performance.

Each input neuron (feature) $a_1^0, a_2^0 .... a_n^0$ of network traffic packet sample x in the input layer 0 are all connected to different hidden neurons $a_1^1, a_2^1 .... a_z^1$ in the first hidden layer 1. Then, each hidden neuron a^i in a hidden layer i is connected to each hidden neuron a^j in the next hidden layer j, where $j = i+1$. The connection between each neuron a^i to the next neuron a^j has a connection weight W^i.

To activate a neuron a^j in the next layer j, we first aggregate the previous neurons connected to it and their weights as follows:

$$\theta_j = \sum(a^i * W^i + b^i),$$

where a^i is a previous neuron in layer i, W^i is a connection weight, and b^i is bias vector. Then, to obtain the output of each neuron a^j in the next hidden layer j, a rectified linear unit (ReLU) activation function is used.

$$a^j = \begin{cases} 0 & \theta_j < 0 \\ \theta_j & \theta_j \geq 0 \end{cases}$$

The output of each neuron a^i in the last hidden layer for class i is transformed using the softmax function. The softmax activation function inputs a vector of neural network outputs and returns a vector of probabilities for each class label, $\tilde{y}$. The following expression represents the equation for the softmax function:

$$\sigma(a^i) = \frac{e^{a^i}}{\sum_{j=1}^{N} e^{a^j}}$$

Where N is the number of classes in the multi-class classifier, in our case it is 3.

The three modules (M_{01}, M_{02}, M_{12}) forms the MNN model. In the MNN model, the three modules are trained in parallel using an MEC server that is close to the IoT devices.

The overall model construction step is summarised in Algorithm 5.1.

Algorithm 3.1 MNN-based Modules constructor

Input : C_i - list of classes, $i = 1, \ldots, K$
X_i - feature vector of all samples belongs to class C_i,
Output : M_{ij} - Binary-classification Trained module for class C_i against class C_j,

```
for all C_i and C_j , i < j do in parallel
    // label the training set T_ij with C_i label=1 and C_j label=0
    T_ij = {(X_i, 1), (X_j, 0)}
    //Train a module M_ij
    M_ij ← Train (T_ij)
end
```

3.4.2 IoT botnet detector

The detector first collects the traffic, extracts the statistical features, and applies normalisation to the features. Then, it uses the three trained modules to infer the class of the traffic. The output of each trained module is finally combined to obtain the final decision.

Traffic collection, feature extraction and normalisation

The network traffic to be analysed is transmitted from the IoT devices to the edge gateway. Subsequently, the traffic is transmitted (offloaded) to the MEC server. which extracts the statistical features of the collected data to be analysed. Statistical features of network traffic need to be gathered so that they can more precisely reflect the characteristics of IoT botnet behaviour. To begin, the network's packets are grouped together into larger units called flows. Second, the network's flow statistics are derived. This study makes use of two publicly available statistical features: IoT23 [125] and MedBIot [111]. We refer the reader to the methods proposed in [125, 111] to extract statistical characteristics of network traffic. The min-max approach was used to normalise the features.

Since MEC servers are typically located close to IoT devices, it is not necessary for the MEC servers to wait an excessive amount of time for data transmission; thus, collecting and extracting the feature and IoT botnet detection would be in near real-time.

Trained modules inference

Using the MNN architecture, the extracted features are fed in parallel to all trained DNN modules and perform inference to predict IoT botnet behaviour in the MEC server. This allows the MEC server to accomplish three detection tasks (Benign vs. Scan, Benign vs. Scan, and Scan vs. CnC) simultaneously. Even though we have already deployed the parallel design on the MEC server, there is still additional work that has to be done to completely optimise parallel computing within the MEC architecture. The scope of this study in this chapter does not include that work.

Module combination

Afterwards, the MEC server combines the inference results of all modules to make the final prediction and sends an alarm to the administrator that monitors the conditions of IoT devices. For the integration of module outputs, three integration units are applied.

First level combination. First, the inverse unit (INV) is used to obtain the output $f_{10}(x), f_{20}(x), f_{21}(x)$ of the reduced modules (M_{10}, M_{20}, M_{21}) from the output of discrimination function $f_{01}(x), f_{02}(x), f_{12}(x)$ of the trained modules (M_{01}, M_{02}, M_{12}) as follows:

$$f_{10}(x) = INV\, f_{01}(x)$$

$$f_{20}(x) = INV\, f_{02}(x)$$

$$f_{21}(x) = INV\, f_{12}(x)$$

Second level combination. For $\forall i, j \in \{1,2,3\}$ and $i \neq j$, the output of discrimination function $f_{ij}(x)$ obtained from the trained modules M_{ij} that trained using the same training set with C_i labelled by 1, which means:

$$T_{ij} = \{(X_i, 1)\} \cup \{(X_j, 0)\}$$

All $f_{ij}(x)$ should be combined by the MIN unit. Then, the discrimination function $f_i(x)$ of class C_i can be obtained by

$$f_i(x) = MIN_{j=0}^{K} f_{ij}(x), j \neq i$$

Final combination. All discrimination functions $f_i(x)$ of all classes are combined using the MAX unit. Therefore, the final discrimination function is

$$f(x) = MAX_{i=0}^{K} f_i(x)$$

Fig. 3.5 shows the module combination process for a three-class problem.

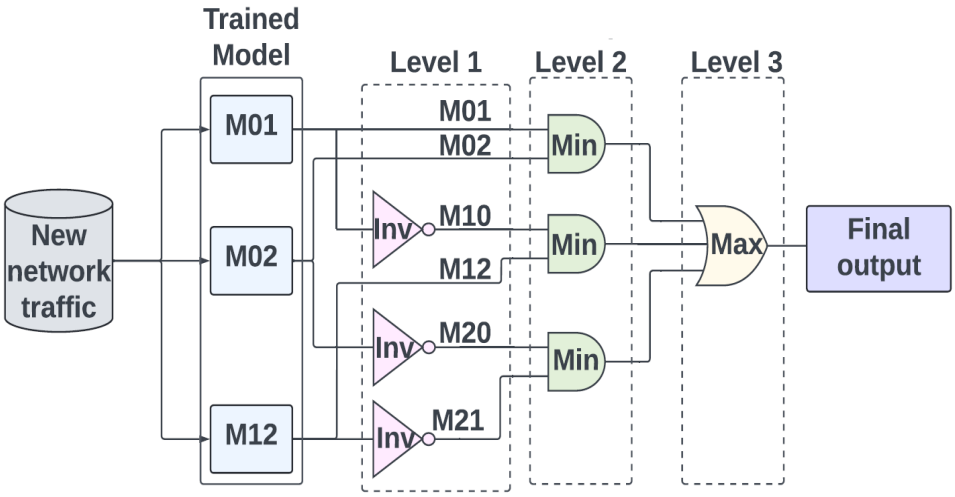

Fig. 3.5 Modules combination for three-class problem during test phase

Comparison Between MNN and One-Vs-One (OvO) multicalssification approch

Both MNN and OvO break the multi-class classification task into binary classifiers, and both approaches allow for parallel training, improving efficiency. However, the key difference lies in how they combine the outputs of these classifiers to make the final prediction. In

MNN, after the binary classifiers produce their outputs, a three-level combination approach is used. First, each classifier generates a probability. Then, the min function is applied to these probabilities in the second level to ensure that only the most confident classifiers are considered, reducing false positives. Finally, the max function is applied to select the class with the highest confidence as the final prediction. This process helps MNN achieve higher accuracy by prioritizing the most reliable predictions and ensuring that the final decision reflects the most confident output.

In contrast, OvO uses majority voting to determine the final class. Each classifier votes for a class, and the class with the most votes is chosen. While this method works well for simpler classification tasks, it does not take into account the confidence level of individual classifiers, which can lead to less accurate predictions when classifiers disagree. As a result, MNN tends to be more accurate, as its multi-level combination approach minimizes false positives and maximizes the reliability of the prediction, while OvO is more prone to errors due to its reliance on majority voting.

Scenario

Let's say we are using a Modular Neural Network (MNN) to classify an input as one of three classes: Benign, Scan, or CnC (Command and Control) in a smart city IoT network. The MNN has three binary-classification modules:

- Module 1 (M01): Benign vs. Scan
- Module 2 (M02): Benign vs. CnC
- Module 3 (M12): Scan vs. CnC

When you feed a network traffic input into this system, each module produces a probability (between 0 and 1) of the input belonging to one of the two classes it is trained to distinguish.

Step 1: Module Outputs

For a given input, the modules give the following probabilities:

- M01 (Benign vs. Scan): The probability of being Benign is 0.7, and the probability of being Scan is 0.3.
- M02 (Benign vs. CnC): The probability of being Benign is 0.6, and the probability of being CnC is 0.4.

- M12 (Scan vs. CnC): The probability of being Scan is 0.8, and the probability of being CnC is 0.2.

Step 2: Using the MIN Function

At the second level of combination, the system applies the MIN function across all outputs to determine the probability of each class (Benign, Scan, and CnC).

For Benign:

$$\text{MIN}(0.7, 0.6) = 0.6$$

For Scan:

$$\text{MIN}(0.3, 0.8) = 0.3$$

For CnC:

$$\text{MIN}(0.4, 0.2) = 0.2$$

Step 3: Using the MAX Function

At the final combination level, the system uses the MAX function to pick the class with the highest probability from the MIN results. The MIN results are:

- Benign: 0.6
- Scan: 0.3
- CnC: 0.2

The system applies the MAX function:

$$\text{MAX}(0.6, 0.3, 0.2) = 0.6$$

Final Output

Since 0.6 is the highest probability, the system classifies the input as Benign.

The MIN function ensures that for each class, the final probability is constrained by the least confident module. For example, even though Module M12 was 80% confident that the input was Scan, Module M01 only gave a 30% chance, so the MIN function chose the lower probability (0.3) to avoid overestimating the likelihood of the input being Scan.

This conservative approach reduces the risk of making false positives by relying on the weakest evidence. By using the MIN function, the system ensures that if any module is not

confident about a particular class, it won't allow that class to be considered strongly in the final decision. This increases the robustness of the classification.

The overall sub-task combination approach is summarised in Algorithm 3.2.

Algorithm 3.2 Early-stage IoT botnet detection (EDIT)

Input :

x - feature vector of IoT traffic
C_i - list of classes, i= 1,..,K
M_{ij} - list of Binary-classification Trained modules

Output :

p_{ij} - M_{ij} module prediction represents the probability of x belongs to C_i against C_j
p_{ji} - M_{ij} module prediction represents the probability of x belongs to C_j against C_i
p_i - probability of x belongs to C_i against all other classes,
P - final predicted class,

// Predict the class of traffic x
for *each* M_{ij} **do**
    $p_{ij} \leftarrow$ `predict`(x)
end
// First level combination of output
for *each pair* (i, j) *where* $i < j$ **do**
    $p_{ji} \leftarrow$ `INV`(p_{ij})
end
// Second level combination of output
for *each class* C_i **do**
    $p_i \leftarrow 1$
    for *each class* C_j *where* $i < j$ **do**
        $p_i \leftarrow$ `MIN`(p_i, p_{ij})
    end
    for *each class* C_j *where* $i > j$ **do**
        $p_i \leftarrow$ `MIN`(p_i, p_{ji})
    end
end
// Final level combination of output
$P \leftarrow 0$
for *each predicted class* p_i **do**
    $P \leftarrow$ `MAX`(P, p_i)
end

3.5 EDIT System Development and Experiments

3.5.1 IoT botnet benchmark datasets

We use two different datasets: IoT-23 [125], MedBIoT [111]. The datasets represent the traffic features during normal, early stages of compromising IoT devices and attacks originated by an IoT botnet. Thus, they are suitable for evaluating the proposed system.

IoT-23. IoT-23 is a dataset that represents the network traffic of IoT devices. Three different real IoT devices have been used to collect this dataset. The dataset contains three sub-datasets of benign network traffic as well as 20 sub-datasets of malicious network traffic. The 20 malicious sub-datasets are collected during the execution of several malicious scenarios related to botnet behaviours. There are a variety of botnet behaviours that were executed during the traffic collection process, such as Mirai, Torii, Trojan, Gagfyt, Kenjrio, Okiru, Hakai, IRCBot, Hajime, Muhstik, Hide and seek. Each botnet consists of five types of behaviours: scanning, command and Control (CnC) , File Download, DDoS, and Heartbeats. The IoT-23 dataset has 20 network traffic features that describe the traffic behaviour during a specific time. Each entry in the dataset is labelled with the type of behaviour: benign or malicious. The type of malicious is also included for malicious entries.

MedBIoT. MedBIoT is a dataset of network traffic features collected from 83 real and emulated IoT devices during normal behaviour (benign samples) and botnet behaviour (malicious samples). Three botnet behaviours were deployed to extract their features, namely, Mirai, Bashlite and Tori. However, only the early stages of the botnet were considered: spreading and CnC communication. The MedBIoT dataset has 100 network traffic features that describe the traffic behaviour during a specific time. The dataset is split into sub-datasets according to the botnet stage and the device type, which makes this dataset appropriate for evaluating MNN.

In this study, besides benign traffic, we only consider the malicious traffic that represents the traffic of the devices infected by Mirai malware from both datasets. The IoT23 dataset has 7 sub-datasets of Mirai malware. These 7 sub-sets were combined into one dataset. We mainly focus on the early phases: scanning and CnC. Therefore, we only consider the flow that is labelled with these two phases. Regarding the MedBIoT dataset, the samples from the sub-datasets of the Mirai botnet (benign, scanning, CnC) were considered. Thus, we built three sub-datasets (DS_0, DS_1, DS_2) from each dataset (IoT23 and MedBIoT), where DS_0 is the dataset of benign traffic, DS_1 is the dataset of scanning behviour traffic, and DS_2 is the dataset of CnC behviour traffic. The EDIT is evaluated using the IoT-23 and MedBIoT separately.

3.5.2 Data pre-processing: normalisation, splitting, partitioning, and labelling

The two datasets display a notable imbalance. Within the realm of cybersecurity, when analysing network traffic, it is customary for the majority of traffic samples to fall under the normal category, whereas attack traffic types are regarded as the minority [127]. Deep learning methods provide less accurate results when classifying attacks on imbalanced datasets [128]. To mitigate the problem of class imbalance, we employ the random under-sampling strategy. Although random under-sampling may lead to the loss of important information from regular traffic, it is acceptable to assume that there are redundant observations in the normal traffic sample. By selectively removing redundant observations, it is improbable that the overall data distribution will see significant alterations. This approach is frequently employed in practical applications because of its straightforwardness and capacity to expedite the learning process.

min-max normalisation was performed, ensuring that no one feature dominates the model training stage. Each sub-dataset (DS_0, DS_1, DS_2) was then split into a training set (90%) and a 10% test (named TE0, TE1, and TE2). The three test sets are then combined into one test dataset that will be used for testing the detector component of the EDIT system.

From the datasets (DS_0, DS_1, DS_2), the 3 training datasets (T_{01}, T_{02}, T_{12}) for Benign against Scan, Benign against CnC and Scan against CnC, respectively, of the three modules (M_{01}, M_{02}, M_{12}), are obtained as described in task decomposition section (section 3.4).

Further, to avoid model over-fitting during training the three modules, 20% of 80% of each set of (T_{01}, T_{02}, T_{12}) were considered as a validation set, producing three validation sets: (V_{01}, V_{02}, V_{12}).

The number of samples per training, validation, and test set is shown in Table 3.4. Whereas Table 3.5 shows the number of samples per network traffic type.

Table 3.4 Number of samples per training, validation and test sets.

Dataset	Training set	Validation set	Test set
IoT23	32422	8105	4506
MedBIoT	63010	15752	8754

Table 3.5 Number of samples per network traffic type.

Dataset	Benign	Scan	CnC
IoT23	15011	15011	15011
MedBIoT	29172	29172	29172

3.5.3 Experimental settings

The EDIT system is evaluated using a MEC server with a 1.70 GHz, 4-core Intel i5 CPU. The experiment was implemented with Python 3.9.12 using TensorFlow 2.10.0 and Keras 2.10.0. We followed our proposed system architecture, first by constructing the model and then by using the model for IoT botnet detection. In our experiment, we used the trained model to test the detection by making predictions on the test dataset.

For the IoT23 dataset, we first constructed three multi-layer perceptron neural network models with Keras. Each model has four layers. The first layer is a dense layer with 20 neurons, and it specifies 20 inputs for the module. The second layer is also a dense layer with 10 neurons. The activation function applied by these two layers is the rectified linear activation function (ReLU). After these layers, a dropout layer is used with a rate of 0.2. This means that during each epoch of training, 20% of the neurons are randomly discarded. The output layer is also a dense layer with two neurons. The output layer used the softmax activation function to produce two outputs. The first output is the probability that each sample (x) belongs to the class labelled with 1 against the class labelled with 0. The second output represents the inversion of the first output. We then compiled the model with the Adam optimizer and the categorical cross-entropy function loss, and we chose the accuracy metric to evaluate the module during training. The modules were trained with 10 epochs and 256 batch sizes, and the validation dataset was used to evaluate each epoch. After constructing the modules, we used these modules to make predictions on the test data. We used the two outputs of each module and combined them using the Min function, and then we used the Max function; however, we used numpy.argmax() to convert the output into class labels.

For the MedBIoT dataset, we did the same procedure. However, the modules' input was 100, and the number of neurons was 100 and 50 for the first and second layers, respectively. The specific parameters are shown in Table 3.6.

Evaluation Metrics

For evaluation, Root Mean Squared Error (RMSE) is used to assess the overall detection performance of the system. RMSE is chosen because it evaluates how well the predicted probabilities align with the actual class labels, providing insight into how confident the model is in its predictions. RMSE is calculated as:

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

Where:

- y_i is the actual class label (either 0 or 1 in a one-hot encoded format, where 1 represents the correct class and 0 represents the other class),

Table 3.6 The parameters of NN modules.

Parameter	IoT23	MedBIoT
First layer (FL)	Dense layer	
Second layer (SL)	Dense layer	
Number of neurons (FL)	20 neurons	100 neurons
Number of neurons (SL)	10 neurons	50 neurons
Activation fun.	ReLU	
Third layer	Dropout layer	
Rate	0.2	
Output layer	Dense layer	
Number of output	2	
Output activation fun.	softmax	
Optimizer	Adam	
Function loss	crossentropy	
Epochs	10	
Batch size	256	

- $\hat{y}_i$ is the predicted probability of the actual class (ranging from 0 to 1),

- n is the total number of samples.

Additionally, a confusion matrix is used to determine the numbers of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). These values are fundamental for calculating several important performance metrics that help evaluate the classification system.

The first metric used is the True Positive Rate (TPR), also known as Recall. This measures the proportion of actual positives correctly identified by the system. It is calculated as:

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

Where:

- TP is the number of true positives (correctly identified positive instances),

- FN is the number of false negatives (missed positive instances).

This metric is crucial for understanding how well the system identifies positive cases, which is particularly important in tasks such as detecting malicious behavior in IoT networks.

The next metric is the True Negative Rate (TNR), or Specificity, which measures the proportion of actual negatives correctly identified by the system. It is calculated as:

$$TNR = \frac{TN}{TN + FP}$$

Where:

- TN is the number of true negatives (correctly identified negative instances),

- FP is the number of false positives (incorrectly flagged benign instances).

Specificity is important for understanding how well the system avoids incorrectly classifying benign traffic as malicious.

Another key metric is the False Positive Rate (FPR), which measures the proportion of actual negatives incorrectly identified as positives. It is calculated as:

$$FPR = \frac{FP}{TN + FP}$$

This metric is important because a high FPR means the system is incorrectly flagging benign traffic as malicious. Reducing FPR is crucial for maintaining the credibility of the detection system.

The False Negative Rate (FNR) measures the proportion of actual positives that were incorrectly identified as negatives. It is calculated as:

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

FNR is critical for understanding how often the system misses actual botnet traffic, potentially allowing malicious activities to go undetected. Minimizing FNR is essential in preventing false negatives in botnet detection.

Accuracy (ACC) is another important metric, measuring the overall correctness of the system in predicting both positive and negative classes. It is calculated as:

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

Tt is used alongside other metrics to provide a more comprehensive evaluation.

The F-Score (or F1-Score) combines precision and recall to give a balanced measure of the system's performance. It is calculated as:

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

Where:

- Precision is the proportion of true positives out of all predicted positives, calculated as:

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

- Recall is the true positive rate, as defined earlier.

The F1-score is particularly useful when the model needs to balance false positives and false negatives. It is especially important when distinguishing botnet behaviors, as it ensures

that both high precision (minimizing false positives) and high recall (detecting as many real threats as possible) are properly balanced.

These metrics were chosen because they provide a comprehensive evaluation of the system's performance across multiple aspects, including accuracy, precision, recall, and how well the predicted probabilities align with the true labels. Each of these metrics contributes to a well-rounded understanding of the system's ability to correctly classify IoT network traffic, particularly when distinguishing between benign and malicious behaviors like botnet activities. The main difference between RMSE and other metrics is that RMSE evaluates the magnitude of the error in the predicted probabilities, giving more weight to larger errors. It measures how well the predicted probabilities align with the true class labels, especially when these probabilities are far from the actual class, which can help highlight misclassifications where the model was confident but incorrect. This is especially useful in this research, where the model is producing probabilistic outputs in the context of IoT botnet detection, where the exact predicted probability matters. Other metrics like accuracy and F1-score are typically calculated after the final output combination (using argmax), which selects the most likely class based on the highest probability. These metrics assess classification correctness, but RMSE provides an additional perspective on the confidence and calibration of those predictions, particularly in distinguishing subtle botnet behaviours in the context of IoT networks.

3.6 Experimental Results

3.6.1 Effectiveness evaluation

Overall detection performance

The Root Mean Squared Error (RMSE) is a crucial metric for evaluating the detection performance of a system, particularly in terms of how well the predicted probabilities align with the true class labels. Table 3.7 presents the average RMSE values obtained from running the IoT botnet detection component of the system. As shown, the RMSE values are close to 0, indicating that the system's predictions are closely aligned with the true labels, reflecting minimal prediction error. The IoT botnet detection component was executed 10 times using different datasets in this study. For each execution, a different value of the random_state parameter was used to randomly split the training and test datasets. The optimal random_state value, which resulted in the smallest prediction errors with an RMSE close to 0, was found to be 715. This value was used to obtain the subsequent evaluation results presented in the following sections.

Table 3.7 Detection Performance (Root Mean Square Error).

	Random state										
Dataset	1	437	984	42	5	66	715	30	16	618	Average
IoT23	0.01	0.00	0.01	0.01	0.01	0.00	0.00	0.01	0.01	0.00	0.006
MedBIoT	0.02	0.01	0.00	0.01	0.00	0.01	0.00	0.00	0.01	0.01	0.007

Comparison with monolithic DNN

The model is evaluated against the monolithic DNN in terms of the performance of classification on the unseen test set. The Multilayer Perception (MLP) was applied to train the monolithic DNN using 100 input neurons, three hidden layers, and three outputs. The rectified linear activation function (ReLU) is used for the hidden layers, and the Softmax activation function is used for the output layer.

The monolithic DNN confusion matrix and MNN confusion matrix are shown in Figures 3.6 and 3.7 for the IoT23 and MedBIoT datasets, respectively.

From the results in Figures 3.6 and 3.7, it is clear that MNN produced zero false positives and zero false negatives.

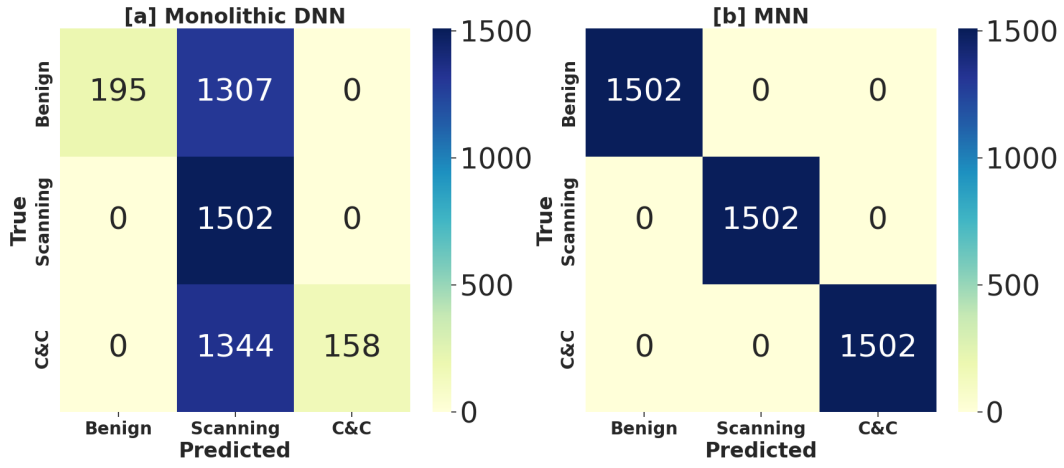

Fig. 3.6 Confusion matrix for Botnet detection in IoT23 dataset

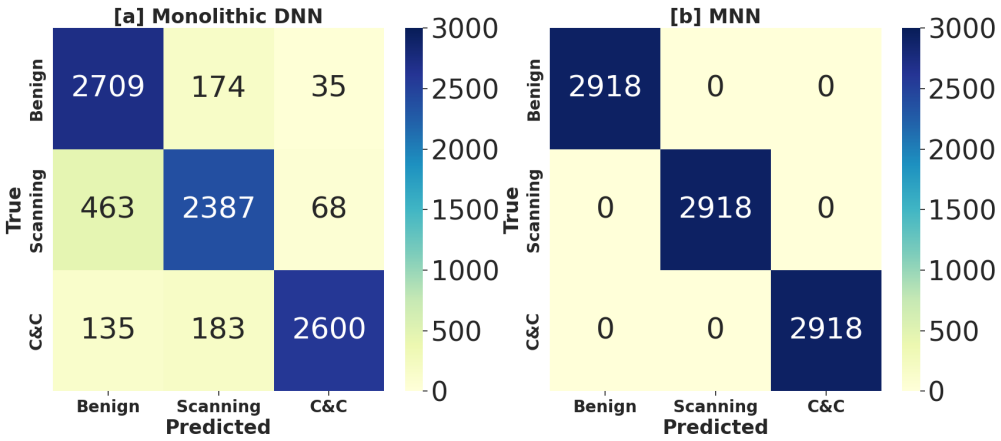

Fig. 3.7 Confusion matrix for Botnet detection in MedBIoT dataset

A summary of the performance of MNN compared to monolithic DNN on performance metrics Precision, recall, and F-1 score are shown in Table 3.8 for IoT23 and MedBIoT datasets.

Based on the experimental findings, it is clear that our suggested MNN attained an accuracy of 100% for both types of early-stage IoT botnet behaviour (scanning and CnC), which was much higher than the accuracy produced by a monolithic DNN in both datasets. It has therefore been shown that the suggested MNN technique to identify an IoT botnet is a highly successful technique.

Table 3.8 Metric results for Botnet detection using IoT23 and MedIoT datasets.

IoT23	Bening		Scanning		CnC	
Metrics	Monolithic	MNN	Monolithic	MNN	Monolithic	MNN
TPR	0.1062	1.0000	1.0000	1.0000	0.1032	1.0000
TNR	0.9996	1.0000	0.1051	1.0000	1.0000	1.0000
FPR	0.0003	0.0000	0.8949	0.0000	0.0000	0.0000
FNR	0.8938	0.0000	0.0000	0.0000	0.8968	0.0000
ACC	0.7010	1.0000	0.4024	1.0000	0.7010	1.0000
F score	0.1920	1.0000	0.5266	1.0000	0.1871	1.0000
MedIoT	Bening		Scanning		CnC	
Metrics	Monolithic	MNN	Monolithic	MNN	Monolithic	MNN
TRP	0.9296	1.0000	0.8087	1.0000	0.9013	1.0000
TNR	0.8974	1.0000	0.9415	1.0000	0.9796	1.0000
FPR	0.1026	0.0000	0.0585	0.0000	0.0204	0.0000
FNR	0.0704	0.0000	0.1913	0.0000	0.0987	0.0000
ACC	0.9081	1.0000	0.8964	1.0000	0.9540	1.0000
F	0.8707	1.0000	0.8414	1.0000	0.9276	1.0000

Comparison with Other ML Algorithms

In this section, we present the comparison between MNN and three widely used machine learning (ML) algorithms, namely Decision Tree (DT), Random Forest (RF), and Support Vector Machine (SVM), using the F-score metric to classify the three classes (benign, scanning, and CnC) across the IoT23 and MedBIoT datasets.

Regarding the hyperparameters used for each classifier, Decision Tree (DT) employs parameters like max_depth, min_samples_split, and min_samples_leaf to control model complexity and prevent overfitting. For Random Forest (RF), the parameters n_estimators, max_depth, min_samples_split, and bootstrap provide flexibility in creating diverse and robust models while maintaining generalization and controlling overfitting. In the case of Support Vector Machine (SVM), the kernel, C, and gamma parameters directly influence how the classifier separates data and generalizes to new samples, with the RBF kernel being commonly used for non-linear problems. These hyperparameters were chosen to ensure that the classifiers are both effective in training and efficient in detecting botnet behaviors in IoT networks. The specific values for each of these hyperparameters are shown in Table 3.9.

The results of the comparison of F-scores across the MNN and ML models are shown in Table 3.10. The MNN model outperforms all other classifiers, achieving perfect F-scores on both datasets. The RF technique performs better than both DT and SVM, despite slight variations in performance across datasets, according to the F-score.

Table 3.9 Hyperparameters used for Decision Tree, Random Forest, and Support Vector Machine classifiers.

Classifier	Hyperparameter	Value Set
Decision Tree (DT)	max_depth	None (default)
	min_samples_split	2 (default)
	min_samples_leaf	1 (default)
Random Forest (RF)	n_estimators	100
	max_depth	None (default)
	min_samples_split	2 (default)
	bootstrap	True
Support Vector Machine (SVM)	kernel	RBF
	C	1.0
	gamma	scale (default)

Table 3.10 Comparison of EDIT (our work) with other ML algorithms.

Dataset	Model	Benign	Scanning	CnC
IoT23	DT	0.90	0.87	1.00
	RF	0.90	0.87	1.00
	SVM	0.69	0.77	0.54
	EDIT(our)	1.00	1.00	1.00
MedBIoT	DT	0.98	0.99	0.99
	RF	0.99	0.99	1.00
	SVM	0.53	0.10	0.34
	EDIT(our)	1.00	1.00	1.00

Comparison with other studies

Previous research that examined botnet detection using a variety of ML/DL algorithms is used as a comparison for the solution that is being suggested here for both the benchmark dataset and the underlying algorithms. As can be seen in Table 3.11, the EDIT performs well, with a false positive rate and a false negative rate that are entirely absent. Compared to earlier research, Table 3.11 also shows that the EDIT provides a better level of classifier performance in terms of accuracy and F score.

3.6.2 Efficiency evaluation

We evaluate the efficiency of EDIT by considering two performance metrics: time and memory consumption for both system components (model construction and IoT bot detection). The metric time is the time in seconds the EDIT takes to construct the MNN model and the time it takes to finish the inference task for the type of one flow traffic in the test dataset.

Table 3.11 Comparison of EDIT (our work) with other studies.

Dataset	Ref	Year	Acc	F-Score	FPR	FNR
IoT23	[107]	2020	0.85	-	0.60	-
	[109]	2021	-	0.94	-	-
	EDIT(our)	-	1.00	1.00	0.00	0.00
MedBIoT	[112]	2021	0.99	0.98	-	-
	[114]	2022	0.99	-	-	-
	EDIT(our)	-	1.00	1.00	0.00	0.00

Whereas memory consumption refers to the amount of storage needed for storing EDIT code and the constructed model that contains three NN modules.

In our experiment, we use a multi-access edge computing server (MEC) with a 4-core Intel i5 CPU. Taking advantage of the modularity of MNN used in EDIT, the three modules can be trained and executed for detection using multi-core parallelism on MEC.

The efficiency performance of EDIT on MEC using the two datasets is shown in Table 3.12. The findings lead to a system that is realistic in terms of model construction and detection time, as well as requiring less storage space.

Table 3.12 Efficiency performance of EDIT on MEC.

IoT23 Dataset		
Metrics	Model construction	IoT bot detection
Time	24.26 secs	0.016326 secs
Storage	5.23 KB	2.52 KB
Model size	123 KB	–
MedBIoT Dataset		
Metrics	Model construction	IoT bot detection
Time	28.90 secs	0.016372 secs
Storage	5.06 KB	2.53 KB
Model size	630 KB	–

3.7 Disscusion

3.7.1 Scalability of the EDIT System

The Modular Neural Network (MNN) architecture of the EDIT system is highly scalable due to its ability to divide the detection task into smaller, specialized modules. Each module can be processed independently, enabling the system to leverage parallel processing across multiple cores or GPUs. This significantly reduces training times and improves performance,

especially when handling large datasets. By distributing the data across multiple processors, substantial performance gains are achieved, particularly for large-scale IoT networks.

The modular design also facilitates easy extension. New detection modules or behaviors can be added without major changes to the existing framework, ensuring that the system can easily adapt to new types of botnet behaviors or evolving IoT traffic patterns.

When it comes to output combination, the system is designed to minimize communication overhead. Since each module operates independently and only exchanges small output results for combination, this process doesn't become a bottleneck. The modular parallelism allows smaller models to be trained across multiple nodes, which is particularly beneficial for running models on resource-constrained devices, such as edge devices or distributed systems.

In environments with limited computational power, such as IoT networks, the MNN's ability to process tasks in parallel across edge devices ensures that the computational load is distributed rather than centralized. This decentralized approach reduces the dependency on continuous communication with a central server, enhancing response time and improving the system's efficiency for real-time botnet detection.

However, several factors could affect scalability. Network communication overhead is one concern, as traffic needs to be sent to all nodes holding specialized modules. If data transfer between nodes isn't optimized, communication latency and bandwidth limitations could become bottlenecks, particularly in large-scale deployments or geographically dispersed systems. Furthermore, as the system processes traffic in parallel, synchronization between modules could introduce delays. Asynchronous processing can minimize these delays by allowing modules to work independently and send results when they are ready.

3.7.2 Robustness of the EDIT System

The robustness of the EDIT system is critical, particularly in real-world environments where network traffic is often unpredictable and subject to various types of attacks or disturbances. In such conditions, the system must maintain its performance and continue detecting threats despite noisy or irregular data.

The system's ability to accurately detect early-stage botnet activities, such as scanning or command-and-control (CnC) behaviors, is enhanced by the multi-level output combination strategy employed by MNN. This strategy ensures that the system remains resilient to noisy data or irregular traffic. By combining the outputs of multiple specialized modules, the system minimizes the impact of weak or unreliable classifiers, focusing on the most confident predictions. This approach helps maintain high accuracy even when faced with traffic that is difficult to detect or resembles benign activities.

Furthermore, the ability to minimize false positives and false negatives is another key factor in the system's robustness. The min function applied during the second-level combination ensures that only the most confident modules are considered for the final decision, which helps reduce misclassifications. This is especially important in IoT networks, where incorrect detection of benign traffic as malicious (false positives) or failing to detect botnet activity (false negatives) can have serious consequences.

However, fault tolerance remains a drawback of the system. In large-scale deployments across multiple devices, failures at specific nodes could disrupt the detection process. Unlike systems that incorporate redundancy or failover mechanisms, the EDIT system does not yet fully address node failure resilience. This lack of redundancy could degrade performance if a node fails during the detection process. Addressing this limitation in future iterations could further improve the system's robustness.

3.7.3 Justification of Perfect Prediction Accuracy

The claim of 100% perfect prediction in the EDIT system's performance is remarkable, but several factors help justify this result. First, the datasets used—IoT23 and MedBIoT—are well-preprocessed and of high quality, making them suitable for training models with high accuracy. These datasets, which are widely used in IoT botnet detection research, are particularly valuable due to their large-scale nature and comprehensive collection of real-world IoT network traffic, ensuring that the model learns from a wide variety of behaviors.

Additionally, the EDIT system underwent thorough evaluation on unseen test data and through cross-validation, with the learning curves showing no signs of overfitting. This indicates that the model generalizes well and is not overly tailored to the training data. The modular structure of the MNN also helps ensure the model's capability to detect distinct botnet behaviors, which improves overall accuracy.

While the perfect prediction rate is impressive, it should be noted that overfitting was checked and not detected. Further real-world testing or deployment in dynamic, diverse environments would provide additional validation. The high accuracy achieved could be influenced by the controlled nature of the evaluation and the careful selection of datasets. In real-world scenarios, where traffic patterns can vary significantly, performance may fluctuate, and further evaluation across different environments will be needed to confirm the robustness of the system.

3.8 Summary

In this research, we propose EDIT, a system designed to detect and investigate communication traffic faults at smart city networks brought on by IoT botnets in their early stages. Multi-access edge computing (MEC) servers located near IoT devices are used to implement EDIT's Modular Neural Network (MNN) technique for IoT botnet detection. As a result of MEC's parallel processing capabilities, modular neural networks (MNN) may improve detection accuracy and efficiency. The results show that the false-negative rate for EDIT is much lower than that of the monolithic method and other studies. When using the MEC server, EDIT model construction consumes little storage space and might take as little as 24 seconds. The result of the EDIT detection task also showed a low detection time of 16 ms. For future work, it might be interesting to investigate the deployment of software-defined networks (SDN) at edge computing to mitigate the IoT botnet at early stages before any intended attack could happen.

Chapter 4

Security Assessment of Hierarchical Federated Deep Learning

Hierarchical federated learning (HFL) is a promising distributed deep learning model training paradigm, but it has crucial security concerns arising from adversarial attacks. This research *investigates and assesses the security of HFL* using a novel methodology by focusing on its resilience against inference-time and training-time adversarial attacks. Through a series of extensive experiments across diverse datasets and attack scenarios, we uncover that HFL demonstrates robustness against untargeted training-time attacks due to its hierarchical structure. However, targeted attacks, particularly backdoor attacks, exploit this architecture, especially when malicious clients are positioned in the overlapping coverage areas of edge servers. Consequently, HFL shows a dual nature in its resilience, showcasing its capability to recover from attacks thanks to its hierarchical aggregation that strengthens its suitability for adversarial training, thereby reinforcing its resistance against inference-time attacks. These insights underscore the necessity for balanced security strategies in HFL systems, leveraging their inherent strengths while effectively mitigating vulnerabilities.

4.1 Context

Federated Learning (FL) offers a promising solution to the challenges of Centralised Machine Learning (CML), including data storage, computation, and privacy. FL facilitates collaborative training of a global model across numerous clients while preserving data decentralisation. This approach has been successful in various applications like smart cities. Traditionally, FL employed a two-level node design, where chosen clients submit updates to a central server, situated either at the *edge* or in the *cloud*, for aggregation, as shown in Fig 4.1(a). The

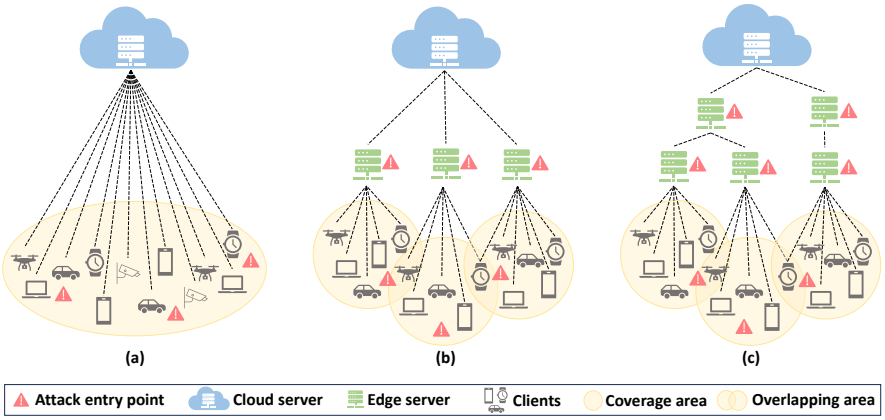

Fig. 4.1 FL network architectures: (a) 2-level FL; (b) 3-level HFL; (c) 4-level HFL

aggregation at the edge improves latency and network efficiency but restricts server capacity, affecting training. The aggregation in the cloud boosts computational power and scalability but may delay updates for distant devices, stressing networks. In recent years, hierarchical federated learning (HFL), a variant of FL, has gained attention. HFL addresses FL challenges by employing multiple aggregator servers at edge and cloud levels, hierarchically interconnected, capitalising on cloud coverage, and reducing the edge server latency [129].

In a use case scenario where HFL is deployed for smart city applications such as image classification, various clients, including smart cars, smart watches, drones, and mobile phones, are scattered across the smart city [130]. A significant number of edge servers are typically deployed in close proximity to these clients, forming a distributed architecture network connected to a central cloud server. Clients establish connections with edge servers within their coverage areas, with overlapping coverage enabling connections to multiple edge servers [131, 132]. These edge servers forward client updates to regional edge servers, ultimately reaching the cloud server for aggregation to build a global model. Fig. 4.1 shows a comparison of 2-level FL (Fig. 4.1(a)) and HFL architectures that can be employed as 3-level [21] (Fig. 4.1 (b)) and 4 level node design [23] (Fig. 4.1(c)).

Despite the advantages of HFL over FL, HFL remains susceptible to adversarial attacks that compromise data integrity by manipulating local datasets or model updates to undermine the global model's performance [133]. In HFL architecture, the increased number of nodes, including clients and edge servers, expands the attack surface, providing more potential entry points for attacks. This amplifies the risk of compromises by malicious edge servers or clients, surpassing the attack surface of FL. Fig. 4.1 provides an overview of the attack surface (see red triangle) in conventional FL compared to HFL. However, the augmentation of nodes also presents opportunities for bolstering defense mechanisms against attacks. This

prompts an exploration of the following question: *How does the HFL architecture impact the robustness of HFL against attacks*?

In recent years, significant attention has been devoted to studying the impact of attacks on FL. Abyane et al. [134] conducted an empirical investigation to comprehensively understand the quality and challenges associated with state-of-the-art FL algorithms in the presence of attacks and faults. Shejwalkar et al. [48] systematically categorised various threat models, types of data poisoning, and adversary characteristics in FL, assessing the effectiveness of these threat models against basic defense measures. Bhagoji et al. [50] explored the emergence of model poisoning, a novel risk in FL, distinct from conventional data poisoning.

In contrast to conventional 2-level FL, adopting HFL introduces many novel research concerns due to its inherently intricate multi-level design [129]. A few studies have focused on examining convergence in HFL [21, 135]. Some studies offer solutions to some of the issues related to HFL security. Zhou et al. [23] introduced a robust model aggregation technique aimed at ensuring the resilience of 4-level HFL against poisoning attacks, particularly in the context of the Internet of Vehicles (IoV). Al-Maslamani et al. [136] tackled the issue of selecting unreliable clients within the 3-level HFL framework to optimise overall HFL security. To the best of our knowledge, scholarly works assessing the security aspects of HFL are relatively scarce. In comparison to these studies, our research focuses on conducting a systematic assessment of the security of HFL. With the growing use of HFL in smart city applications [133], it is crucial to evaluate their resilience and understand their architectural nuances to suggest areas for improvement.

This chapter explores how the HFL architecture withstands adversarial data injected during inference. Our findings highlight the challenges inference-time attacks pose to model accuracy. Yet, defense strategies like adversarial training offer promising solutions. We delve into Data Poisoning Attacks (DPA) and Model Poisoning Attacks (MPA) at the client and server sides during training, alongside potential defense mechanisms within the HFL framework. We identify vulnerabilities to targeted DPA (backdoor attack), notably in the 4-level HFL model, where hierarchical structure affects malicious client selection probabilities. Implementing the neural cleanser method [137] proves effective against targeted backdoor attacks, emphasizing tailored defense strategies' importance. Conversely, HFL models show resilience against untargeted DPA and MPA due to multi-level aggregation, mitigating outlier impact and enabling recovery from attacks.

In summary, our contributions to address Research Question 2 of this thesis are as follows:

1. We propose an experimental framework for HFL security assessment that evaluates its resilience against inference-time attacks through comparative analysis, experimental

validation, and defense mechanism testing. This framework enhances our understanding of HFL's robustness under adversarial conditions.

2. Through comparative analyses, we pinpoint vulnerabilities in HFL under various training-time attacks and investigate how the HFL architecture influences model resilience against attacks, deepening our understanding of FL design and security.

3. Our assessment of adversarial hierarchical federated training via extensive experiments on different datasets and HFL architectures sheds light on effective defense mechanisms for future HFL framework development, emphasizing HFL's resilience and its capacity to recover from attacks.

4.2 Security Assessment of HFL

Hierarchical Federated Learning (HFL) improves scalability but introduces security risks, making it vulnerable to adversarial attacks. This section provides a brief overview of key security aspects relevant to this study. Section Section 4.2.1 outlines the HFL model and its security implications. Section 4.2.2 explores adversarial attacks, including data and model poisoning. Section 4.2.3 discusses defense mechanisms, such as adversarial training and anomaly detection. While Chapter 2 covers these topics in detail, this section highlights key points relevant to our experiments.

4.2.1 Hierarchical Federated Learning (HFL) Model

We conceptualise the HFL system as a multi-parent hierarchical tree (as shown in Fig. 4.1), denoted as $T = (V, E)$, consisting of $|L|$ levels. Nodes in the system, categorised as clients (N) and servers (S), are represented in the set V, while the collection of undirected communication channels between nodes is represented in the set E. The cloud server node, s_0, serves as the root of the tree at level 0, with client nodes, n, positioned at the leaves of the tree at level $L-1$. Intermediate edge servers, s_ℓ, act as intermediary nodes between cloud servers and clients at level ℓ ($\ell \in \{1, \ldots, L-2\}$). Clients may train their local models using local data and transmit their model parameters to regional edge servers s_{L-2} for aggregation. The aggregation process in an HFL system involves several critical steps shown in figure 4.2. (*Step 1*) The cloud server s_0 sends the initial model to clients n through edge servers s_ℓ. (*Step 2*) Regional edge servers s_{L-2} select a set of client participants C_t at aggregation round t from their coverage areas $A(s_{L-2})$ for model updates. (*Step 3*) Clients C_t download the latest model from regional edge servers s_{L-2} and train their local models. (*Step 4*) Updated parameters

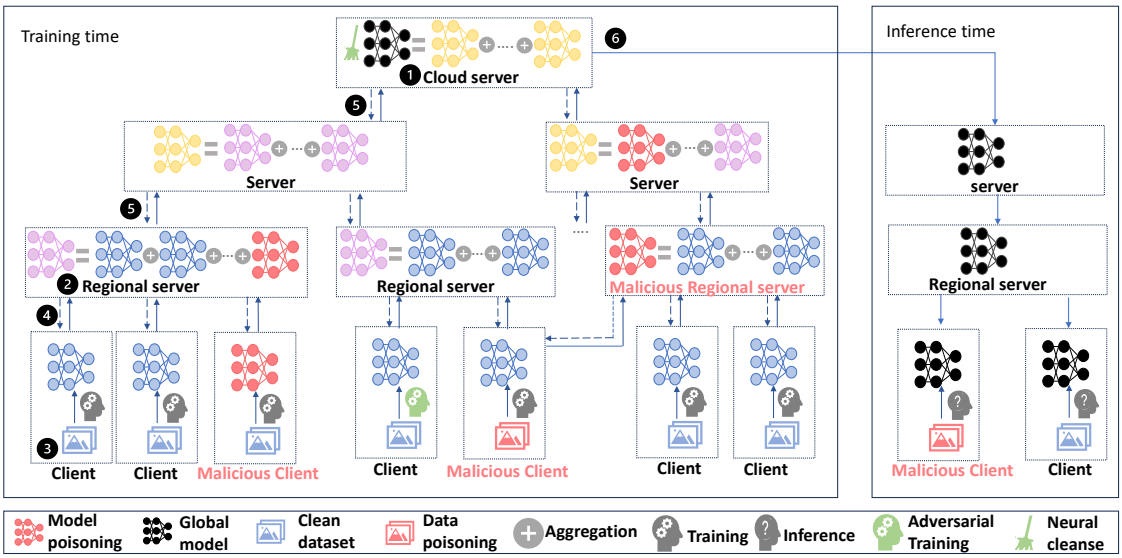

Fig. 4.2 HFL and Attack Model

are sent back to regional edge servers s_{L-2} for aggregation. (*Step 5*) Parent servers s_ℓ at level ℓ aggregate updated model parameters from child nodes $s_\ell + 1$ within their coverage areas $A(s_\ell)$ for T_ℓ number of aggregation rounds. (*Step 6*) After T_0 global aggregation rounds implemented by cloud server s_0, a global model is constructed and transmitted to clients for deployment through edge servers s_ℓ.

We employ the averaging aggregation method proposed by McMahan et al. [138], allowing flexibility in deploying HFL models with varying levels (L).

4.2.2 Adversarial Attacks on HFL Model

We consider the attacks on HFL models targeting data integrity during both the *training* and *inference* time. These attacks can be client-side or server-side, with client-side attacks encompassing data poisoning and model poisoning tactics. More details on these attacks were provided in Chapter 2.

Inference-time Attacks (ITAs). ITAs aim to carefully perturb the input data at inference time to have them misclassified by the global model. Adversarial data is created through two types of ITAs: white-box attacks and black-box attacks, determined by the attacker's access level to the target global model. White-box attacks require full access to the target model, including its architecture, parameters, and gradients. Black-box attacks, on the other hand, do not rely on or require access to the internal details of the target global model. In this work, we have applied white-box attacks, including Adversarial Patch(AP), Fast Gradient

Method(FGM), Projected Gradient Descent(PGD), and Saliency Map Method(JSMA). We also applied black-box attacks, including Square Attack(SA) and Spatial Transformations Attack(ST) [137].

Training-time Attacks (TTAs). TTAs aim to inject adversarial data during training time to influence model parameters. These attacks can be client-side or server-side. Client-side attacks encompass data poisoning attack (DPA) and model poisoning attack (MPA) tactics. On the server side, the attacker can only implement MPA. DPA aims to manipulate the training data, while MPA directly alters model parameters. To implement the DPA attack, we apply the *targeted label flipping (TLF)* method [137], which aims to make the model misclassify specific backdoored inputs and maintain the model performance on the other inputs. We also applied an *untargeted label flipping (ULF)* attack that introduced random misclassifications. Regarding the MP, we implement *client-side sign flipping attacks (CSF)* and *server-side sign flipping attacks (SSF)* that flip the sign of the model parameters. Fig. 4.2 shows the attack models during inference time and training time.

4.2.3 Adversarial Defense on HFL Model

Defenses against adversarial attacks can be broadly classified into two categories: *data-driven* and *model-driven* defenses [90]. Data-driven defenses involve detecting adversarial attacks in the data or enhancing the quality of the data corrupted by the attack to improve the performance of the model. These defense methods are typically agnostic to the learning architecture [90]. Model-driven defenses involve building models that are robust to adversarial attacks.

In this work, to study the architectural impact of HFL on the efficacy of defense methods, we only implement model-driven defense methods that reconstruct the trained model to make it more robust. Thus, we implement *Neural cleanse (NC)* [137], a defense method that cleans the neural network from the neurons that are possibly affected by an attack. This method helps mitigate the impact of a TLF backdoor attack and produces a new, robust model. NC can be applied to the global model on the cloud server before it is sent to the clients. We also implement a well-known defense called *adversarial training (AT)* [38]. AT is the process of retraining the model with adversarial examples to make the model recognise these examples and classify them correctly, even in the presence of perturbations. In the context of HFL, we can call it *adversarial hierarchical federated training*. Each client implements local AT and collaborates with clients during adversarial hierarchical federated training to construct a robust global model against adversarial attacks in inference time.

4.2.4 Experiment Design

We conduct experiments to assess the impact of adversarial attacks on HFL models (3-level HFL and 4-level HFL) and compare the performance of HFL models under various attacks and defense mechanisms alongside CML and traditional FL approaches (2-level FL). Our code is available on GitHub[1]. The experimental settings are summarized as follows:

Dataset. We use three popular image classification datasets: mnist, fashion-mnist, and cifar-10. Each dataset contains 60,000 images (of which 50,000 images are in the training set and 10,000 images are in the test set) categorised into 10 classes. To simulate non-IID real-world scenarios, the images of the training set are split according to the Dirichlet distribution. We use state-of-the-art implementation of attack and defense methods from [137].

HFL model. We consider a population of smart devices representing client nodes distributed across a city that implements image classification tasks. A group of 100 clients exists that engage in communication with the server for the purpose of image classification model training. We assume that the client selected for participation remains constant throughout the training process. Every client trains a local classifier model to classify the images. Regarding the server nodes, there is one cloud server in each learning paradigm at level 0. The cloud server performs the FedAvg aggregation rule for 20 aggregation rounds. We assume that the cloud server is highly secure and has never been compromised during the learning process. On the other hand, edge servers in HFL have different characteristics. In 3L-HFL, there are 20 regional edge servers that are distributed at the same level and connected directly with the cloud server and directly with 5 clients in their coverage area. Each regional edge server performs the FedAvg aggregation rule for two aggregation rounds. The 4L-HFL has similar settings to the 3L-HFL; however, there are 4 edge servers distributed at the same level between the cloud server and the regional edge servers. Each edge server communicates with five regional edge servers and performs the FedAvg aggregation rule for three aggregation rounds. The total aggregation round of regional edge servers is 40 and 120 rounds for 3L-HFL and 4L-HFL, respectively.

Client local training model. We use two different convolutional neural network (CNN) architectures for the client's local classifier model for the three datasets. For mnist and fashion-mnist, we deploy a CNN with two 3x3 convolution layers (the first with 32 channels, the second with 64, each followed by 2x2 max-pooling), a fully connected layer with 512 units and ReLu activation, and a final softmax output layer with 10 outputs. For cifar10, A CNN with two 3x3 convolution layers with 32 channels followed by 2x2 max pooling, another two 3x3 convolution layers with 64 channels followed by 2x2 max pooling, a fully connected layer with 512 units and ReLu activation, and a final softmax output layer with 10

[1]https://github.com/dalqattan/SecHFL

outputs. Each client employs categorical cross-entropy as their loss function and utilises the optimiser that implements the Adam algorithm to update their local model depending on the loss function. For the mnist and fashion-mnist dataset, the batch size was set to 32 and the number of epochs was set to 1. For the cifar10 datasets, the batch size was set to 64, and the number of epochs was set to 6. Table 4.1 summarizes the CNN model architectures used for training on the MNIST, Fashion-MNIST, and CIFAR-10 datasets.

Attribute	**MNIST & Fashion-MNIST**	**CIFAR-10**
Convolution Layers	2x (3x3, 32 & 64 channels) + 2x2 max-pooling	2x (3x3, 32 channels) + 2x2 max-pooling, 2x (3x3, 64 channels) + 2x2 max-pooling
Fully Connected Layer	512 units, ReLU activation	512 units, ReLU activation
Output Layer	Softmax, 10 outputs	Softmax, 10 outputs
Loss Function	Categorical Cross-Entropy	Categorical Cross-Entropy
Optimizer	Adam	Adam
Batch Size	32	64
Epochs	1	6

Table 4.1 Summary of CNN Model Architectures

Malicious Node. If a client is compromised, the client could act maliciously by implementing DPA or MPA. We evaluate the performance of the model while the number of malicious clients is 1, 5, and 10. We also evaluate the models when all of the malicious clients are located in the overlapping area of two regional edge servers. We indicate the model that considers the overlapping area with the letter 'O' (3-level HFL-O and 4-level HFL-O). We also assume that regional edge servers can be compromised and act maliciously by implementing MPA, whereas other edge servers are highly secure. We evaluate the performance of the model while the number of malicious servers is 1, 5, and 10.

Evaluation Metrics. We include the Misclassification Rate (MR) and the Targeted Attack Success Rate (TASR) to assess attack efficiency and defense effectiveness. The Misclassification Rate (MR) can be formulated as:

$$MP = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(f(x_i^{'}) \neq y_i), \tag{4.1}$$

where n is a number of image examples, $f(x_i^{'})$ is the aggregated model's output (global model output for HFL or centralized model output for CML) over input $x_i^{'}$ which is clean input x_i for training-time attacks and adversarial input x_i^{adv} for inference-time attacks, y_i is ground

truth, and $\mathbb{I}(\cdot,\cdot)$ is an indicator function that returns 1 if model's output does not match with the ground truth.

Similarly, TASR can be formulated as:

$$TASR = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(f(x_i^{adv}) = y_i^{adv} \mid y_i^{adv} \neq y_i), \tag{4.2}$$

where $f(x_i^{adv})$ is the aggregated model's output over adversarial input $x_i^a dv$ for a targeted adversarial attack label y_i^{adv} in a backdoor attack, and $\mathbb{I}(\cdot,\cdot)$ is an indicator function that returns 1 if model's output on attacked input matches with targeted adversarial attack label.

4.3 Results and Discussion

4.3.1 Baseline performance: HFL model under no attacks

This section compares the performance of four models: a centralised machine learning model (CML), a 2-level FL, a 3-level HFL, and a 4-level HFL. As shown in Fig. 4.3, the CML model maintains consistently high accuracy across 20 global aggregation rounds over each dataset. The 4-level HFL model demonstrates notably high performance, showcasing the potential advantages of hierarchical architecture in FL. The 3-level HFL model presents an intermediary performance between 4-level HFL and 2-level HFL models, showing how hierarchical architecture impacts FL. HFL architecture enhances model update efficiency and potentially leads to faster convergence. In contrast, the 2-level FL model shows inferior performance.

The improved accuracy of HFL models can be attributed to several key factors:

- Multi-level aggregation reduces noise in model updates by refining them at intermediate levels before reaching the global model, leading to more stable and representative global updates.
- More hierarchy levels allow for localized aggregation, which helps balance non-IID (heterogeneous) data distribution among clients. This prevents extreme variations in updates and improves convergence.
- HFL introduces additional aggregation rounds at different hierarchy levels (edge, regional, and global). More frequent aggregations help smooth out model updates more quickly, enabling the model to converge faster than a flat 2-level FL structure with fewer aggregation steps.

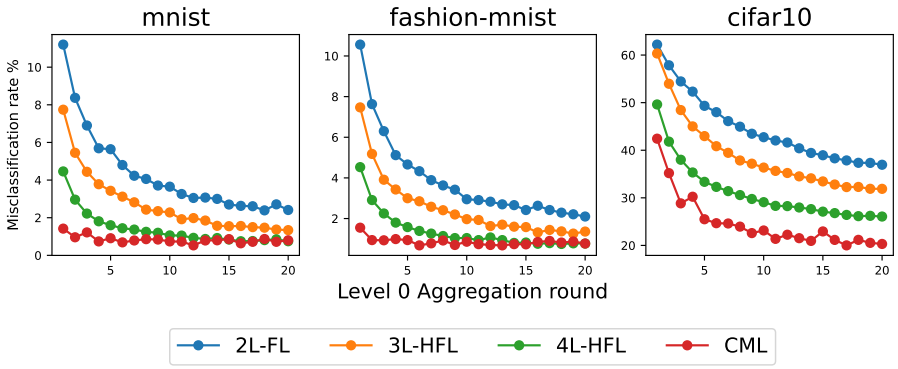

Fig. 4.3 Baseline performance: HFL models performance without adversarial attacks.

4.3.2 Models performance under Inference-time attacks and defense

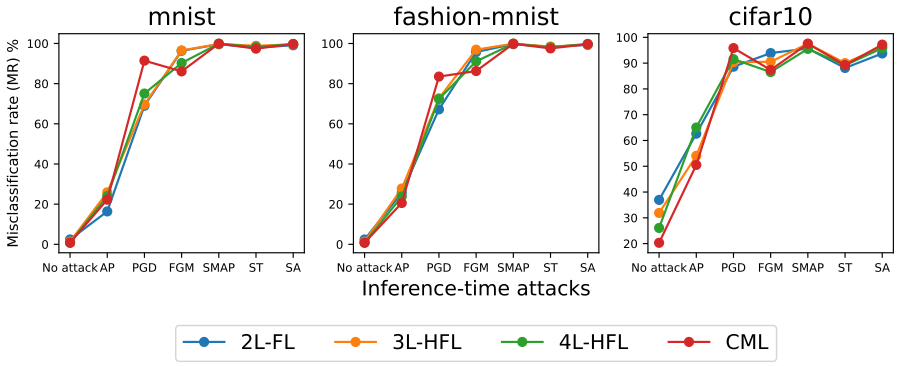

Fig. 4.4 Models performance under inference-time adversarial attacks.

Impact of the attacks. We assessed the effectiveness of the models under attack by calculating MR. The outcomes are presented in Figure 4.4. Upon analysing the MR, it becomes evident that the MR of all models trained on the same dataset exhibits a high degree of similarity. However, the impact of each type of attack can vary. Adversarial patch attacks demonstrate the lowest impact. All the other attacks lead to a high MR ranging between 80% to 100%. As highlighted in [139], in cases when the models, optimisation methods, and the poisoned test dataset are identical, the effects of attacks on accuracy are likely to be comparable for both Centralised machine learning and federated learning models. However, the reason for studying the impact of inference-time attacks in HFL is that many defenses against inference-time attacks are implemented during training. Thus, it is crucial to study the architectural impact on the model's robustness against inference-time attacks.

Adversarial training (AT) defense against inference-time attack. We adversarially trained all models using data generated by inference-time attacks to enhance their robustness. The effectiveness of these adversarially trained models was evaluated by measuring MR, as shown in Table 4.2. In general, the MR dropped significantly across all models. While adversarially trained FL models demonstrate comparable MR to CML models, HFL models, especially the 4-level architecture, show even lower MR, suggesting higher resistance to attacks.

Table 4.2 Robustness of models (performance as per minimizing MR) due to AT (defense). The number in bold is the best defense among FL architectures

Dataset	Model	AP	PGD	FGSM	JSMA	ST	SA	Average
mnist	2L FL	8.82	2.27	2.97	2.05	2.66	4.93	3.95
	3L HFL	15.64	1.48	2.13	1.89	11.87	7.34	6.73
	4L HFL	**5.35**	**0.92**	**1.6**	**0.97**	**1.15**	**1.8**	**1.96**
	CML	5.16	0.96	1.71	0.84	6.73	2.13	2.92
fashion-mnist	2L FL	12.58	2.31	2.88	2.56	2.22	4.98	4.59
	3L HFL	8.29	1.32	1.74	1.59	**1.68**	3.6	3.04
	4L HFL	**5.65**	**0.9**	**1.27**	**1.07**	6.58	**2.13**	**2.93**
	CML	5.86	1.06	1.91	1.01	9.52	2.75	3.69
cifar10	2L FL	44.28	49.67	43.1	41.14	**60.42**	51.92	48.42
	3L HFL	39.95	47.51	39.05	37.53	56.17	43.2	43.90
	4L HFL	**34.89**	**41.79**	**38.15**	**32.16**	62.75	**39.81**	**41.59**
	CML	28.56	27.35	30.17	22.36	60.94	29.23	33.10

Fig. 4.5 shows the improved MR of robust models (red solid line) achieved through adversarial training compared to vulnerable models (red dashed line). However, a drawback of direct adversarial training adoption is observed with increased dataset complexity (cifar10), leading to higher MR for clean data, emphasising the need for further research on complex, large-scale datasets.

4.3.3 Models performance under Training-time attacks and defense

Fig. 4.6 shows the consequences of training-time attacks on five distinct FL models that possess varied degrees of hierarchy and compromised nodes across different clean test datasets. The x-axis shows the number of compromised nodes (0, 1, 5, and 10), while the y-axis signifies the impact of the attack, reflecting the increase in MR resulting from the training-time attacks. The letter 'O' in the model name indicates that all the malicious clients are located in an overlapping area of two regional servers.

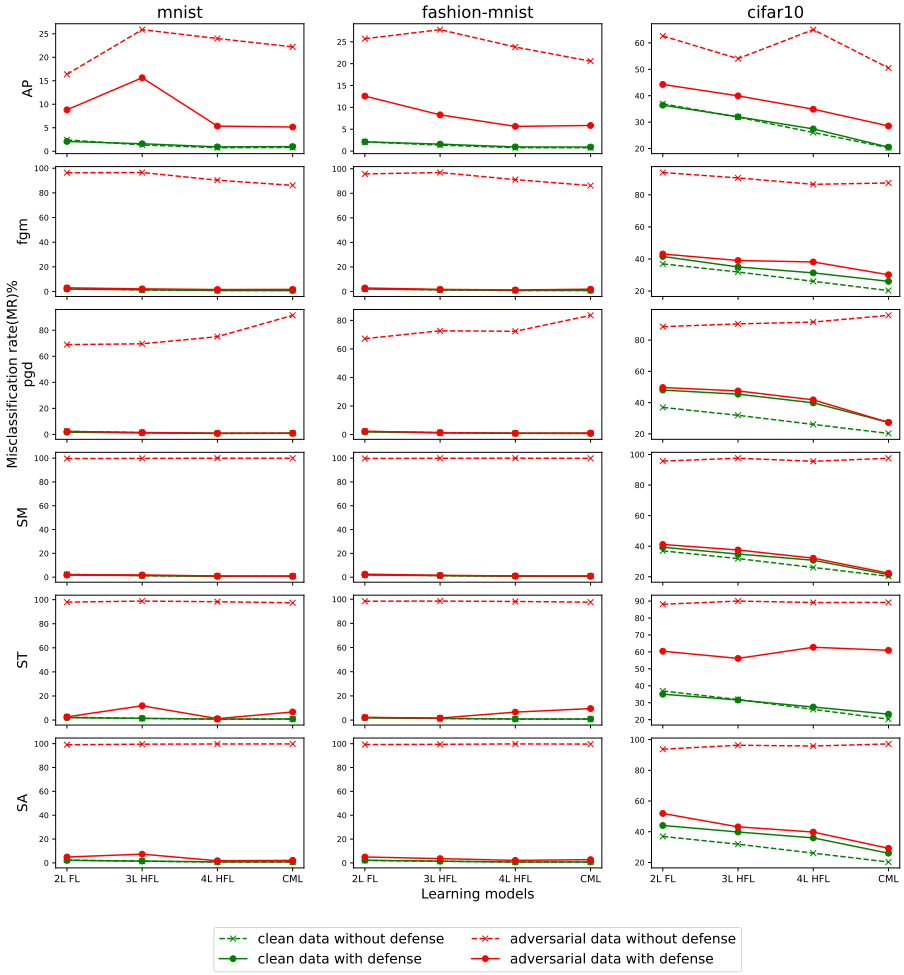

Fig. 4.5 Model's performance under Inference-time attacks and adversarial Training defense

The impact of client-side attacks (data poisoning)

We study both targeted and untargeted attacks on HFL as follows:

Targeted label flipping (TLF) with backdoor attack. The targeted backdoor attack has two aims. First, to maintain the model's performance on clean data. Second, to make the model misclassify the targeted label as a desired label.

TLF backdoor attack result in Fig. 4.6 shows that the MR for all three clean test datasets remains relatively stable across different percentages of malicious clients. This stability suggests that the presence of malicious clients has little impact on the model's performance, even when malicious clients are located in the overlapping areas of two servers. This indicates that the attacker fully achieved the first aim of not influencing the model's performance on clean test datasets.

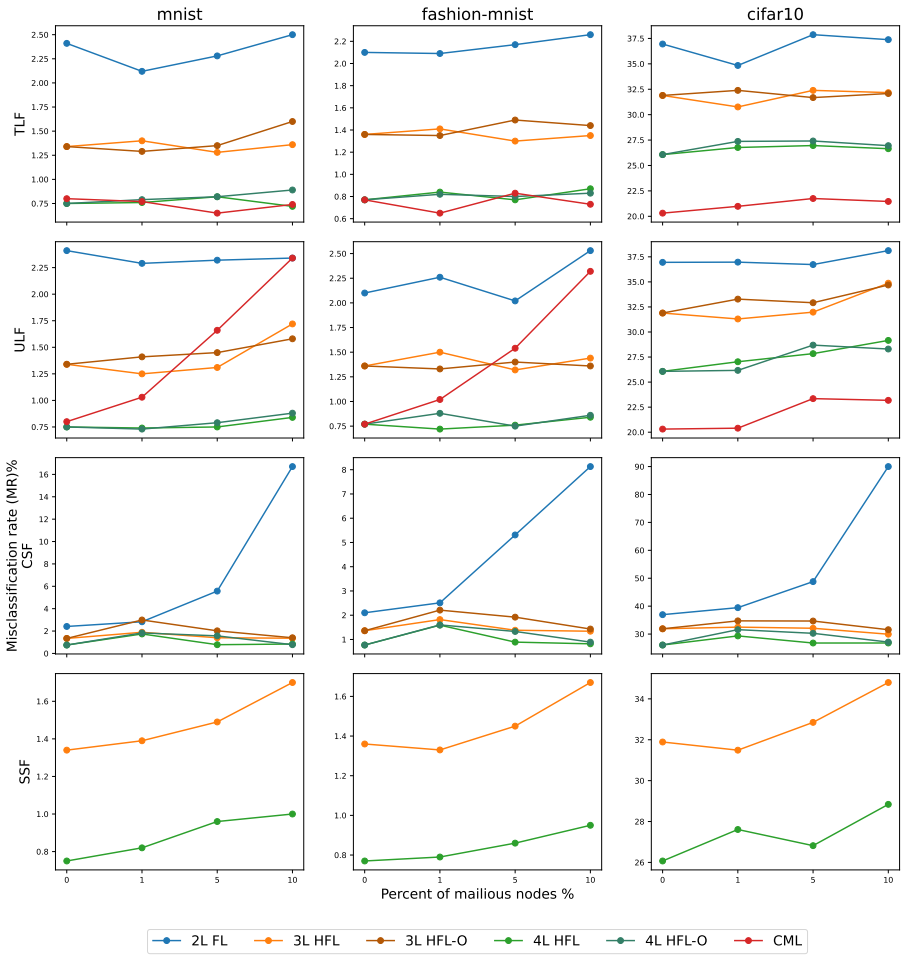

Fig. 4.6 Model's performance under Training-time attacks

The analysis of the second aim is shown in Fig. 4.7. From Fig. 4.7, we observe that TASR increases with the percentage of malicious clients for all models. CML model shows a notably high TASR, indicating vulnerability to backdoor attacks. Among FL models, the 4-level model consistently demonstrates the highest vulnerability to backdoor attacks, followed by the 3-level model and then the 2-level model. This suggests that increased complexity in FL models does not necessarily correlate with improved security against backdoor attacks. Malicious clients located in the overlapping area further amplify the potency of backdoor attacks, underscoring the importance of tailored security measures required in FL environments.

The neural cleanse (NC) method offers a robust defense against backdoor attacks in FL models, significantly reducing TASR and enhancing overall model security and robustness. Fig. 4.7 (solid lines) shows the effectiveness of this method across various FL models, show-

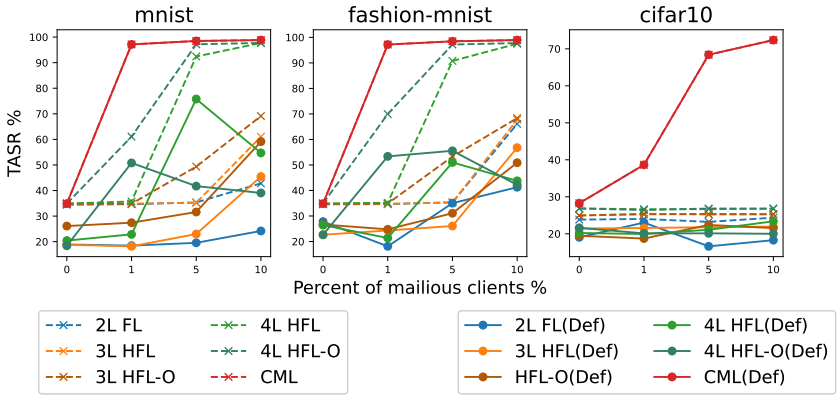

Fig. 4.7 Success rate of backdoor attacks before (dashed line) and after (solid line) neural cleanser defence.

casing a substantial reduction in TASR compared to scenarios without defense mechanisms (dashed line).

Despite these improvements, CML models still show higher TASR values, highlighting their inherent vulnerabilities to backdoor attacks compared to FL models. This underscores the inherent vulnerabilities of CML systems to backdoor attacks and emphasises the relative resilience of FL models when equipped with NC defense mechanisms. The degree of improvement in TASR varies based on factors like dataset complexity, percentage of malicious clients, and model architecture, emphasising the need for adaptive defense strategies tailored to specific attack scenarios.

Untargeted random label flipping (ULF) attack. As shown in Fig. 4.6, CML models suffer amplified effects from such attacks as increased compromised clients. However, FL and HFL are less impacted. For instance, in the mnist dataset, with 10 compromised clients, the MR increases by only 0.2% compared to models without attacks. Although HFL has slightly higher susceptibility due to server coverage, its impact remains minimal. FL's resilience is attributed to its client selection mechanism, where only a small proportion of clients are chosen per round, reducing the likelihood of selecting compromised clients. Moreover, to reduce FL and HFL accuracy, more than 10 clients must be compromised, necessitating a high-budget attack. Furthermore, imposing constraints on local dataset sizes effectively mitigates the occurrence of poisoned data, offering an efficient defense against untargeted attacks. This observation is consistent with findings presented in [48], further supporting the resilience of FL in real-world scenarios.

Impact of client-side attacks (model poisoning)

In model poisoning [Client-side Sign flipping (CSF)], we only evaluate the result for FL models. This is because model poisoning is not commonly applied in CML. Regarding model poisoning attacks, Fig. 4.6 shows that all five FL models show minimal increases in MR, indicating resilience against such attacks. However, the 2-level FL model displays significant vulnerability when 10 clients are compromised, as observed in [48]. Conversely, the 3-level and 4-level HFL models show stronger performance, attributed to their hierarchical aggregation process, which mitigates the impact of individual clients. Even when all compromised clients strategically overlap two servers, HFL models show lesser MR impact compared to the 2-level model. These findings underscore the importance of hierarchical structure in mitigating model poisoning effects, suggesting the need for enhanced security measures for the 2-level FL model.

The impact of server-side attacks(model poisoning)

In comparing server-side sign-flipping (SSF) attacks between 3-level and 4-level HFL models, we observe in Fig. 4.6 that the 4-level model consistently shows lower MR across all datasets, indicating greater resilience to model poisoning. The impact increases with the number of compromised servers yet remains negligible, with both models showing only a slight increase in MR even when 10 servers are compromised. Specifically, the MR increase for the 3-level model does not exceed 0.4% for mnist and fashion-mnist datasets, while for CIFAR-10, both models show only a 3%-4% increase in MR. These results highlight the robustness of HFL models against server-side attacks, particularly for the 4-level architecture.

From the results of a systematic analysis of HFL security, we observe that, in the context of ITAs, HFL models show varying degrees of susceptibility to adversarial perturbations during the inference phase. These findings underscore the importance of evaluating model robustness against a diverse range of ITAs to ensure reliable performance in real-world scenarios. AT emerges as a promising defense strategy, effectively enhancing model robustness against such attacks. Notably, adversarially trained FL models, especially those HFL models, demonstrate competitive misclassification rates compared to CML. The 4-level HFL architecture, in particular, shows notable resilience in adversarial training, suggesting its efficacy in mitigating adversarial attacks.

Regarding TTAs, the 4-level HFL model shows the highest vulnerability to TLF attack, particularly when malicious clients are positioned in overlapping areas of regional servers. However, our investigation also assesses the effectiveness of defense mechanisms, such as

the NC method, in mitigating TLF attacks within HFL systems. The NC method significantly reduces the TASR, enhancing the overall security posture of HFL models.

Moreover, FL and HFL models show greater resilience to ULF attacks, with minimal MR increases even when a considerable number of clients are compromised. This resilience can be attributed to the multi-level aggregation inherent in HFL, which effectively smooths out the impact of outliers introduced by such attacks. This ability to recover from attacks further underscores the robustness of HFL in real-world deployment scenarios.

4.4 Summary

Our experiments demonstrate that Hierarchical Federated Learning (HFL) exhibits resilience against untargeted data poisoning attacks, as its multi-level aggregation process helps mitigate the impact of corrupted updates before they reach the global model. However, targeted attacks, particularly backdoor attacks, achieve a higher success rate in HFL compared to standard FL, especially when malicious clients position themselves within the overlapping coverage areas of regional edge servers. This finding indicates that the hierarchical structure, while improving robustness against random attacks, also introduces vulnerabilities that adversaries can exploit to insert hidden triggers into the global model more effectively.

To assess HFL's security under different attack scenarios, we conducted extensive experiments on data poisoning attacks (DPA), model poisoning attacks (MPA), and inference-time adversarial attacks across various HFL architectures. Our results reveal that higher levels of hierarchy improve robustness against untargeted attacks but increase susceptibility to backdoor attacks, as hierarchical aggregation enables malicious updates to propagate more effectively through multiple layers before reaching the global model. The experiments also show that the placement of malicious clients significantly influences attack success, with overlapping edge server coverage areas being the most vulnerable zones for adversarial exploitation.

In addition, our adversarial training experiments demonstrate that HFL can enhance defense against inference-time attacks, such as adversarial examples, by leveraging its multi-tier aggregation structure. However, while adversarial training improves robustness, it also slows down model convergence, suggesting a trade-off between security and efficiency. Our findings indicate that more frequent aggregation rounds in HFL contribute to faster convergence, but they also amplify the effects of targeted attacks, requiring additional security mechanisms.

These experimental insights highlight the need for HFL-specific defense mechanisms to mitigate targeted attacks at different levels of aggregation. Future work should focus

on adaptive aggregation strategies, enhanced backdoor detection, and real-time adversarial monitoring to ensure the security and reliability of HFL in distributed learning environments

Chapter 5

Dynamics of Hierarchical Federated Learning Under Attacks

Hierarchical Federated Learning (HFL) offers a scalable solution for distributed deep learning across client-edge-cloud systems, improving model training efficiency while preserving data privacy. However, HFL systems are vulnerable to model discrepancies caused by data heterogeneity and malicious attacks, such as data and model poisoning, which can hinder the convergence of the global model. This study investigates the dynamics of model discrepancy in HFL architectures under various attack scenarios and high data heterogeneity. We introduce a Model Discrepancy score that integrates multiple metrics—Dissimilarity, Distance, Uncorrelation, and Divergence—to measure discrepancies in model updates and detect malicious activity. Our experimental results demonstrate that while the Model Discrepancy score effectively distinguishes between benign and malicious edge servers, the consistency within each group varies. Factors such as data heterogeneity, attack sophistication, and the complexity of the hierarchical architecture can lead to divergence even among benign servers, reducing their expected model similarity. Likewise, malicious servers may show high internal consistency depending on the attack strategy. In distributed attack scenarios, the 4-level HFL architecture struggles to maintain clustering quality, as overlapping discrepancy patterns between benign and malicious servers complicate differentiation due to additional aggregation level. These findings underscore the need for robust aggregation techniques that account for these influencing factors, highlighting the limitations of current methods.

5.1 Context

Federated learning has emerged as a promising approach for collaborative deep learning in distributed environments, offering significant advantages over traditional centralised training models [139]. It enables decentralised model training while preserving the privacy of user data and reducing communication costs, which are crucial in such environments [140]. The core algorithm involves training local deep learning models independently on multiple clients and periodically averaging the model parameters on a global server [138].

Despite its advantages, federated learning introduces model discrepancy across clients. The data distribution among clients is often non-IID ((non-Independent and Identically Distributed) in nature, meaning that local training steps can lead to significant divergence in the learned models across the parameter space [141]. This divergence can impede the convergence of the global model, particularly in non-IID settings, as averaging a large number of divergent local models can negatively impact global loss minimisation [142]. Additionally, federated learning systems are vulnerable to a range of security threats. Malicious actors can carry out attacks such as data poisoning and model poisoning, compromising both privacy and stability [143]. Data poisoning attacks involve malicious clients manipulating their training data by introducing noise or altering target labels, which results in a poisoned model. Similarly, in model poisoning attacks, malicious clients directly modify their models before submitting them to the server. These attacks manipulate the model parameters, creating substantial discrepancies between malicious and benign client updates.

Consequently, evaluating model updates before aggregation and quantifying the discrepancies among clients have emerged as effective approaches to defend against such poisoning attacks. Detecting and filtering out suspicious updates is essential to mitigate their impact on federated learning systems [144]. Model discrepancy-based robust aggregation has been proposed as a promising defense mechanism, aiming to detect and reject malicious updates by analysing metrics such as dissimilarity [145, 146] or distance [147] between client updates before aggregation.

Recent advancements in Federated Learning (FL) have led to the emergence of Hierarchical Federated Learning (HFL), a variant that addresses some limitations of traditional FL [130]. HFL structures the learning process across a client-edge-cloud continuum, organising nodes into a hierarchy with multiple aggregation levels, including edge servers, regional servers, and a global server [20]. This multi-level aggregation approach enhances scalability and efficiency in data collection and model training [148]. However, the added complexity of multiple aggregation layers in HFL introduces challenges that change the dynamics of model training compared to traditional FL [133].

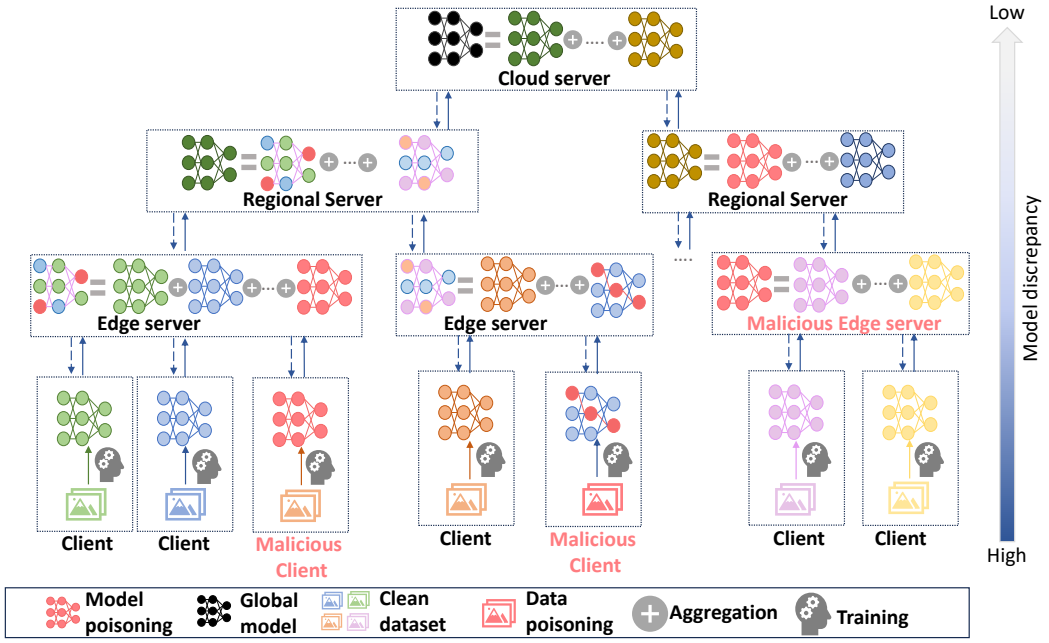

Fig. 5.1 Illustration of Hierarchical Federated Learning (HFL) architecture. The figure depicts the client, edge, and cloud levels within the HFL framework. At the client level, each client is represented with datasets of different colours, symbolising the data heterogeneity across clients. Arrows indicate the flow of model updates from the client level to the edge servers and then to the cloud, highlighting the model discrepancy degree, which decreases as updates are aggregated upwards from the heterogeneous client datasets. Attacks, particularly at the client or edge level, can contribute to changes in model discrepancy degree, affecting the overall model integrity as updates propagate through the architecture.

While HFL offers advantages in resource efficiency and scalability, it remains susceptible to model discrepancies caused by data heterogeneity and poisoning attacks, potentially hindering global model convergence. Like traditional FL, HFL is vulnerable to data and model poisoning at the client level, but it also faces the additional threat of model poisoning at the server level [41]. Therefore, model discrepancy-based robust aggregation techniques from traditional FL need to be adapted to handle the unique characteristics and complexities of HFL. Figure 5.1 illustrates Hierarchical Federated Learning (HFL) architecture, highlighting the model discrepancy degree, which decreases as updates are aggregated upwards from the heterogeneous client datasets. Attacks, particularly at the client or edge level, can contribute to changes in model discrepancy degree, affecting the overall model integrity as updates propagate through the architecture.

Ongoing research [41, 23, 42] focuses on designing discrepancy-based robust aggregation methods tailored to HFL to mitigate the impact of poisoning attacks, as summarised in Section 5.2. However, current model discrepancy-based robust aggregation methods face several

limitations. They often overlook the high heterogeneity of client data, rely on a single metric to calculate the discrepancy between the client and global models or between pairwise client models, assume that benign clients' models are more similar (i.e., less discrepant) than those of malicious clients, and typically consider only targeted or untargeted attack scenarios. As HFL continues to expand its application to large-scale edge devices in the client-edge-cloud continuum, its complexity due to intermediate aggregation presents new challenges that are not encountered in traditional FL. Therefore, understanding the discrepancy phenomena in HFL under attack is crucial for designing robust discrepancy-based aggregation methods tailored to HFL.

To address these challenges, we conduct a comparative experimental study that provides in-depth insights into the dynamics of two well-known HFL architectures (3-level HFL and 4-level HFL) by analysing the discrepancy phenomena caused by various attack scenarios in the presence of high data heterogeneity. Our study highlights the limitations of state-of-the-art robust aggregation methods in HFL and suggests directions for future research.

In summary, our contributions to address Research Question 3 of this thesis are as follows:

- We develop and validate a Model Discrepancy score that integrates multiple metrics, including Dissimilarity, Distance, Uncorrelation, and Divergence. This score provides a robust measure of discrepancies in model updates, enhancing the detection of malicious activities in hierarchical federated learning (HFL) systems.
- We conduct a comprehensive experimental study comparing the dynamics of two well-known HFL architectures (3-level HFL and 4-level HFL) under various attack scenarios with high data heterogeneity. This analysis provides valuable insights into the discrepancy phenomena in HFL and highlights the limitations of current robust aggregation methods.
- We identify several factors that significantly influence model discrepancy in HFL systems, including data heterogeneity, attack sophistication, aggregation layers, and the metrics used for discrepancy measurement. Understanding these factors is crucial for developing more effective, robust aggregation techniques tailored to the complexities of HFL architectures.

The rest of this chapter is organised as follows: Section 5.2 reviews the related work in federated learning and hierarchical federated learning under adversarial attacks. Section 5.3 details the methodology, including the architectures, datasets, attack models, and analysis methods used in this study. Section 5.4 presents the results and analysis, focusing on temporal and spatial dynamics, and clustering effectiveness under attack scenarios. Section

5.5 discusses factors Affecting Model Discrepancy. Finally, Section 5.6 concludes the paper, summarising the key contributions and insights gained from this research.

5.2 Related work

In Section 2.4.2, we highlight a series of robust defense algorithms that is proposed to determine Model Discrepancy between benign and malicious models by using similarity-based aggregation methods, as surveyed in [96]. In this section, we provide a detailed discussion of the related work that aligns with the objectives of this chapter.

Poisoning attacks, which are related to attacking the model training phase, have attracted extensive attention. Researchers have been focused on the study of data poisoning attacks and local model poisoning attacks. By poisoning the training data, data poisoning attacks aim to corrupt the trained model, which has been demonstrated to be effectively harmful in many FL systems. Malicious clients may corrupt the global model by adding, modifying, and/or deleting examples in the local training datasets. In addition, malicious clients may launch model poisoning attacks, which manipulate the model updates transferred to the server[149].

Researchers design defenses based on various metrics to evaluate the local models before aggregation, assuming that the poisoned local model is significantly different from the benign one. Commonly used metrics include similarity metrics (e.g.,Manhattan distance [150], and cosine similarity[151]). For example, FLTrust [145] a federated learning framework that utilises a combination of discrepancy-based and trust-based mechanisms to defend against federated learning poisoning attacks. FLTrust computes a trust score for each client's model update based on its similarity to a reference update and a reputation score. Another framework is ARFED (Attack-Resistant Federated Averaging), which proposes an outlier elimination mechanism to mitigate the impact of data and model poisoning attacks. ARFED uses a discrepancy-based method to detect and remove malicious updates before aggregating the global model[146].

While these studies deployed a single detector for the server in the 2-level architecture to develop defensive mechanisms for traditional federated learning (FL), they are not directly applicable to the Hierarchical Federated Learning (HFL) system due to several reasons. Firstly, HFL introduces additional complexity with its multi-level aggregation process, involving edge servers, regional servers, and a global server. This hierarchical structure means that discrepancies and attacks can propagate through multiple layers, requiring a more nuanced and multi-layered detection approach. Secondly, the increased number of aggregation points in HFL systems means that a single detector at the global server level is insufficient to capture the discrepancies that might occur at intermediate levels. Lastly, the

diverse and potentially heterogeneous data distributions at various levels of the hierarchy necessitate tailored detection mechanisms that can handle the specific characteristics and challenges at each level, which a single detector designed for traditional FL is not equipped to manage. Therefore, while single-detector mechanisms work for simpler FL architectures, they fail to address the intricacies and layered nature of HFL systems.

There are few defences that are based on Model discrepancy metrics to mitigate against poisoning attacks in HFL. Zhou et al. [23] propose a robust 4-level HFL framework named RoHFL for the Internet of Vehicles(IoV), which allows HFL to be suitably applied in IoV systems with robustness against poisoning attacks. The authors develop a robust model aggregation scheme, that contains a logarithm-based normalisation mechanism to cope with scaled gradients from malicious vehicles. The cosine similarity is measured between aggregated gradient at cellular base stations (CBSs) with local model of vehicles and aggregated model of road side units (RSUs) to evaluate the trustworthiness of vehicles and road side units (RSUs). In this way, the global model can eliminate the compromised RSU updates and contaminated vehicle submissions for global aggregation based on trustworthiness score. The research study by Zhou et al.[42] discusses the use of a detection mechanism called Partial Cosine Similarity (PCS) to detect and mitigate adverse behaviours in 3-level HFL system. It can detect anomalous behaviours at the edge by comparing the difference between the previous edge model and the currently uploaded client model. The author claim that client can normally participate in training round if the compared model is higher than specified threshold ; otherwise, the client is dropped to protect model-aggregation. Hence, the edge detector checks all local models uploaded from clients are checked and then trusted subset of clients are returned to the server. Thereafter, the edge server aggregates local models from filtered clients. Another study by Zhao et al. [41] proposes a flexible defense approach called "FlexibleFL" to mitigate poisoning attacks in 3-level HFL systems. The key idea of FlexibleFL is to evaluate the quality of the uploaded model parameters and determine the contribution of participants through an optimal threshold selection strategy. In the context of HFL, a higher similarity might indicate that the models are learning similar features from the data, which could be a sign of healthy, non-malicious participation. The FlexibleFL approach involves setting a threshold for cosine similarity. Model parameters that fall below this threshold in terms of similarity to others might be flagged or given less weight in the aggregation process. Specifically, the paper uses cosine similarity to measure the differences between the weight parameter features of each of the two participants.

In the context of HFL which has different learning process from FL, these studies only use one of Model Discrepancy metric to detect the deviation of aggregated client or edge server models from the previous aggregated model to filter out the suspicious ones. In

contrast, we study Model Discrepancy as a function of different metrics to evaluate the Model Discrepancy between edge servers during learning process in terms of four characteristics namely, dissimilarity, distance, uncorrelation and divergence. These metrics calculated by cosine similarity, euclidean distance, Pearson Correlation and Jensen-Shannon Divergence, respectively. Our aim is to evaluate the impact of different attacks on the dynamics of HFL learning process through the Model Discrepancy.

5.3 Hierarchical Federated Learning Architectures

Based on graph theory,we conceptualise a HFL system as a multi-parent hierarchical tree denoted as $T = (V, E)$, consisting of $|L|$ levels donated as $\ell \in 0, 1, \ldots, L-1$. The set V represents the nodes in the system, which can be categorised into two types: clients (N) and servers (S). The set E represents the collection of undirected communication channels that exist between the nodes. The global server node, denoted as s_0, serves as the root of the tree and is located at level 0. The client nodes, represented as n, are situated at the leaves of the tree and are positioned at level $L-1$. Edge servers, denoted as s_ℓ, act as intermediary nodes positioned between global servers and clients at level ℓ. A coverage area $A(s_\ell)$ is associated with each edge server. Our primary focus lies in examining the model discrepancy of edge servers at level $L-2$, specifically those that serve as parents to client nodes.

We deploy the averaging aggregation method as proposed by McMahan et. al. in [138]. Different from the algorithm in [21] which proposes 3-level FL to model HFL processes, our algorithm can be flexibly deployed for L-level FL, where $L \in \{2, 3, 4...\}$. When $L = 2$, the training is traditional FL. Whereas when $L > 2$, the training is considered HFL. The details of the L-level FL procedure are presented in Algorithm 5.1. The key notations used in Algorithm 5.1 are summarised in Table 5.1.

This study evaluates two hierarchical federated learning architectures: the 3-Level Hierarchical Federated Learning (3LHFL) and the 4-Level Hierarchical Federated Learning (4LHFL). Each architecture represents a distinct structure, characterised by different layers and configurations. The aggregation procedure in a HFL system is comprised of several critical steps depend of the hierarchical levels, which are outlined for 3LHFL and 4LHFL as follows:

5.3.1 3-Level Hierarchical Federated Learning (3LHFL)

The 3LHFL architecture consists of three hierarchical layers: clients, edge servers, and a global server. The key components and interactions in this architecture are:

Table 5.1 List of key notation

Symbol	Description
$\|L\|$	Number of levels in the hierarchy
ℓ	A level in the hierarchy where $\ell \in 0, 1, \ldots, L-1$
N	Set of clients
$\|N\|$	Number of clients
n	A client where $n \in N$
C_p	Proportion of client participants
C_t	Set of client participants selected at aggregation round t
D_n	Dataset of client n where $n \in N$
$\|D_n\|$	Number of sample in Dataset of client n where $n \in N$
S	Set of servers
$\|S\|$	Number of servers
s_ℓ	A server at level ℓ where $s \in S$
s_0	The global server
s_{L-2}	A regional edge server at level $L-2$ communicate with clients
$A(s_\ell)$	Set of nodes in coverage areas A of server $s \in S$ at level ℓ
$\|D_{s_\ell}\|$	Number of sample in Dataset of all clients in $A(s_\ell)$
T_ℓ	Number of aggregation rounds at level ℓ
t	Current aggregation rounds
w_t	Weight vector of the model at round t
$\|Ep\|$	Number of epoch for client local training
B	Batch size for local training
b	A batch of training data set
η	Learning rate
$\Delta F(w;b)$	The average gradient on b of client's local data at the current model w_t

Algorithm 5.1 L-level HiFedAvg

Input : D_n - Dataset of client n where $n \in N$
Output : w - weight vector of the global model

//Run on client n
Function `ClientUpdate(`w`):`
  $\beta \leftarrow$ (split D_n into batches of size B)
  for *local epoch e from* 1 *to* $|Ep|$ **do**
    foreach *batch* $b \in \beta$ **do**
      $w \leftarrow w - \eta \Delta F(w;b)$
    end
  end
  return w
//Run on server s
Function `Aggregation(`s_ℓ,w_t`):`
  foreach *round* $t = 1, 2, \ldots, T_\ell$ **do**
    if $\ell = L-2$ **then**
      //random set of clients
      $C_t \leftarrow$ select randomly C_p % of clients from $A(s_\ell)$
      foreach client $n \in C_t$ **do in parallel**
        $w_{t+1}^n \leftarrow$ ClientUpdate(w_t)
      end
      $w_{t+1} \leftarrow \frac{\sum |D_n| . w_{t+1}^n}{\sum |D_n|}, n \in C_t$
    else
      foreach server $s_{\ell+1} \in A(s_\ell)$ **do in parallel**
        $w_{t+1}^{s_\ell} \leftarrow$ Aggregation($s_{\ell+1}$,w_t)
      end
      $w_{t+1} \leftarrow \frac{\sum |D_{s_\ell}| . w_{t+1}^{s_\ell}}{\sum |D_{s_\ell}|}$
  end
  return w_{t+1}
//Start federated learning
$w_0 \leftarrow$ initialise model's weight
$w \leftarrow$ Aggregation(s_0, w_0)

- **Clients:** The first layer consists of multiple clients (e.g., mobile devices or IoT devices) that generate and preprocess local data. These clients train local models on their respective data and periodically send the model updates to their corresponding edge servers.

- **Edge Servers:** The second layer comprises edge servers that aggregate model updates from the clients within their network. The aggregation process involves averaging the model weights received from the clients. Once the aggregation is complete, the edge servers send the averaged models to the global server. Although, there are no regional server, we consider that a group of edge servers are located in close proximity (i.e they are in the same region)

- **Global Server:** The final layer includes a global server that aggregates the model updates received from the edge servers. The global server performs a final aggregation, averaging the model weights to create a global model. This global model is then disseminated back to the edge servers and, subsequently, to the clients.

5.3.2 4-Level Hierarchical Federated Learning (4LHFL)

The 4LHFL architecture introduces an additional layer, enhancing scalability and organisational efficiency. The components and interactions in this architecture are:

- **Clients:** Similar to the 3LHFL architecture, the first layer consists of clients that generate and preprocess local data. These clients train local models and send their updates to the edge servers.
- **Edge Servers:** The second layer consists of edge servers that aggregate model updates from the clients within their network. The aggregation process involves averaging the model weights received from the clients, reducing the communication burden on the higher layers.
- **Regional Servers:** The third layer introduces regional servers that aggregate model updates from the edge servers in coverage area and form a region. This further reduces the communication burden on the global server by handling part of the aggregation process.
- **Global Server:** The final layer includes a global server that performs the final aggregation of model updates received from the regional servers. The aggregation involves averaging the model weights to create a global model. This global model is then distributed back down the hierarchy to the regional, edge, and client layers.

5.4 Attack Model

In Hierarchical Federated Learning (HFL), adversaries can disrupt the behaviour of deep learning systems by targeting the training process at various levels of the hierarchy. These attacks manipulate the learning process, leading to compromised global models that fail to accurately represent the underlying data or perform as intended. The attacks can be broadly classified into **Data Poisoning Attacks (DPA)** and **Model Poisoning Attacks (MPA)**, depending on whether the adversary manipulates the training data or the model updates [149].

Data Poisoning Attacks (DPA) target the client level, where adversaries manipulate local training data to mislead the global model. Such attacks introduce biased behaviour into the learning process, altering the relationships between features and labels in the dataset [152]. This study evaluates two DPA variations:

- **Targeted Label Flipping (TLF):** In TLF, adversaries deliberately flip specific labels in the training data to create backdoors. This causes the global model to behave abnormally only when encountering particular inputs, allowing the adversary to exploit the model in controlled scenarios while maintaining normal performance otherwise [75].
- **Untargeted Label Flipping (ULF):** In ULF, adversaries flip labels randomly, introducing sporadic disruptions into the learning process. This results in unpredictable global model behaviour as the system struggles to adapt to erratic updates [74].

Model Poisoning Attacks (MPA) manipulate gradients or model updates, altering the aggregation dynamic across the hierarchy. These attacks introduce systematic errors into the learning process, leading to degraded global models [153]. Two MPA types are studied:

- **Client-Side Sign Flipping (CSF):** In CSF, adversaries compromise clients to invert gradient signs, creating behaviour in the learning process that diverges from normal optimisation pathways. This systematically degrades model performance by introducing persistent errors during aggregation [154].
- **Server-Side Sign Flipping (SSF):** SSF occurs when edge or regional servers are compromised to flip the signs of aggregated updates. This propagates erroneous behaviour to higher aggregation levels, disrupting the global learning process and causing long-term degradation [23].

The hierarchical structure of HFL significantly influences how adversarial attacks impact the learning dynamic. A **region** in an HFL architecture is defined as a collection of clients or servers within close proximity, typically grouped based on geographic location or network topology. Regions can be categorised as follows:

- **Benign:** No nodes in the region are compromised, leading to stable and consistent learning dynamic.
- **Malicious:** All nodes in the region are compromised, resulting in highly erratic updates and misaligned learning dynamic.

- **Semi-malicious:** The number of malicious nodes exceeds benign nodes, creating an unstable balance that skews the learning process.
- **Semi-benign:** The number of benign nodes exceeds malicious nodes, but the learning dynamic may still show minor inconsistencies due to some malicious influence.

These classifications provide a framework for understanding how attacks affect the dynamic of deep learning in HFL. By examining discrepancies introduced by compromised nodes, this study evaluates how these attacks influence the integrity and reliability of the learning process across hierarchical levels.

5.5 Model Discrepancy

In previous studies, Model Discrepancy has typically been measured using a single metric. However, we argue that a single metric is insufficient, as different attack behaviors impact model updates in distinct ways. To address this limitation, we introduce the Model Discrepancy Score, which integrates multiple discrepancy metrics to provide a more comprehensive assessment.

Each metric captures a unique aspect of model discrepancy:

- Dissimilarity (Cosine Similarity): Measures angular differences in updates, identifying directional shifts.
- Distance (Euclidean Distance): Captures magnitude differences in model parameters.
- Uncorrelation (Pearson Correlation): Detects hidden dependencies and inconsistencies in updates.
- Divergence (Jensen-Shannon Divergence): Identifies probabilistic shifts in update distributions.

To ensure fair contribution from each metric, we normalize all values to [0,1]. Without normalization, certain metrics (e.g., Euclidean Distance) could dominate, leading to biased results. This normalization ensures that each metric contributes equally to the final Model Discrepancy Score, making it a balanced and interpretable measure.

While deep learning models contain millions of parameters, condensing model discrepancy into a single score does not imply a loss of information. Instead, it provides a high-level summary of model inconsistency, making it practical for monitoring and detecting adversarial behaviors in Hierarchical Federated Learning (HFL). Below, we describe how each

metric is computed and how the overall Model Discrepancy Score is derived, where $\vec{A}$ and $\vec{B}$ represent the weight vectors of two models, A_i and B_i denote the i-th components of $\vec{A}$ and $\vec{B}$, respectively.

5.5.1 Dissimilarity

To measure Dissimilarity, Cosine Similarity is used to measure the cosine of the angle between two non-zero vectors of an inner product space. It is a measure of orientation and not magnitude, ranging from -1 (completely dissimilar) to 1 (completely similar). The formula is:

$$\text{Cosine Similarity} = \frac{\vec{A} \cdot \vec{B}}{||\vec{A}|| \cdot ||\vec{B}||}$$

where $\vec{A}$ and $\vec{B}$ are the vectors representing the model weights. It is normalised as follows:

$$\text{Normalised Cosine Similarity} = 1 - \frac{\text{Cosine Similarity} + 1}{2}$$

A normalised value of 0 indicates perfect similarity, while a normalised value of 1 indicates perfect dissimilarity.

5.5.2 Distance

For this characteristic, Euclidean Distance is used Measures the straight-line distance between two points in Euclidean space. It ranges from 0 (close) to ∞ (distant). The formula is:

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

where A_i and B_i are the components of vectors $\vec{A}$ and $\vec{B}$. It is normalised for our purposes:

$$\text{Normalised Euclidean Distance} = \frac{\text{Euclidean Distance} - \min}{\max - \min}$$

where min is the smallest Euclidean distance observed between any pair of model weight vectors across the nodes, and max is the largest.

A normalised value of 0 indicates no distance (identical vectors), while a normalized value of 1 indicates the maximum observed distance (most dissimilar vectors).

5.5.3 Uncorrelation

Uncorrelation is measured using Pearson Correlation that measure the linear correlation between two vectors. It ranges from -1 (inverse correlation) to 1 (perfect correlation) whereas 0 indicates no correlation between the vectors. The formula is:

$$\text{Pearson Correlation} = \frac{\sum_{i=1}^{n}(A_i - \overline{A})(B_i - \overline{B})}{\sqrt{\sum_{i=1}^{n}(A_i - \overline{A})^2} \cdot \sqrt{\sum_{i=1}^{n}(B_i - \overline{B})^2}}$$

where $\overline{A}$ and $\overline{B}$ are the means of vectors $\vec{A}$ and $\vec{B}$. It is normalised using the absolute value:

$$\text{Normalised Pearson Correlation} = 1 - |\text{Pearson Correlation}|$$

A normalised value of 0 indicates perfect correlation, while a normalised value of 1 indicates no correlation (completely uncorrelated).

5.5.4 Divergence

Jensen-Shannon Divergence used to measure the divergence between two vectors $\vec{A}$ and $\vec{B}$. It ranges from 0 (no divergence) to 1 which corresponds to maximum divergence when the two vector are completely different. The Jensen-Shannon Divergence between two vectors $\vec{A}$ and $\vec{B}$ is defined as:

$$\text{JSD}(\vec{A}||\vec{B}) = \frac{1}{2}\sum_i A_i \log\frac{A_i}{M_i} + \frac{1}{2}\sum_i B_i \log\frac{B_i}{M_i},$$

where $M_i = \frac{1}{2}(A_i + B_i)$.

Here, i denotes an index representing the individual events in two vectors $\vec{A}$, $\vec{B}$ and $\vec{M}$. The summation $\sum_i$ runs over all possible outcomes in the sample space, and A_i, B_i and M_i represent the value i in vectors $\vec{A}$, $\vec{B}$ and $\vec{M}$, respectively.

5.5.5 Model Discrepancy score

The overall Model Discrepancy score is calculated using the Quadratic Mean (also known as the Root Mean Square) of all normalized metrics. This is computed as follows:

$$\text{Overall Model Discrepancy Score} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\text{Normalized Metric}_i)^2}$$

where N is the number of metrics.

The Quadratic Mean (Root Mean Square) is used because it provides a balanced aggregation of multiple metrics while giving more weight to larger discrepancies. This sensitivity to larger values ensures that significant deviations, which are more indicative of malicious activity, are emphasised in the overall score. The Quadratic Mean combines metrics like Dissimilarity, Distance, Uncorrelation, and Divergence in a way that reflects their individual magnitudes without allowing any single metric to disproportionately influence the overall score. This method prevents underestimation of the overall discrepancy score and smooths out minor fluctuations, making it a robust tool for detecting significant anomalies indicative of potential attacks in federated learning systems.

5.5.6 Analysis Methods

Our study inspects the the pairwise model discrepancy score of edge servers from three critical perspectives. (1) Temporal perspective, (2) Spatial perspective and (3) Clustering perspective

Temporal Analysis

To understand the impact of various attacks on the degree of model discrepancy during hierarchical federated learning (HFL), for each aggregation round, we computed the pairwise model discrepancy score between the model updates of each pair of edge servers, quantifying the differences in model parameters. These pairwise discrepancy scores were then averaged across all pairs of edge servers to obtain a single discrepancy score for each aggregation round. This process was repeated for each round to track the temporal evolution of average discrepancy scores. The analysis compared the temporal discrepancy patterns under clean conditions (no attacks) with those under various attack scenarios, providing insights into how attacks and the architecture influence model discrepancy.

Spatial Analysis

To assess the impact of malicious activity on hierarchical federated learning (HFL) systems, we performed a spatial analysis of edge server model discrepancies. We compared pairwise model discrepancy scores across edge servers within the same region (intra-region) and between different regions (inter-region). For each attack scenario, we calculated discrepancy scores among edge servers within a single region and also between edge servers from that region and those from other regions. We visualised model discrepancy distributions using box plots to identify patterns related to the regions' statuses—whether malicious, semi-malicious, semi-benign, or benign.

By focusing on the first aggregation round, when the impact of attacks is most significant, we were able to capture the initial disruptions caused by malicious activity. This analysis provided insights into how the distribution of model discrepancy scores shifted from clean learning conditions to attack conditions, highlighting the effects of different attack scenarios on the overall system.

Clustering Analysis

For the clustering analysis, we aim to evaluate whether model discrepancy scores can be effectively used to distinguish between benign and malicious edge servers. We begin by calculating the pairwise model discrepancies of each edge server with every other server, resulting in a matrix of pairwise model discrepancy scores at the first aggregation round.

To handle the high dimensionality of the discrepancy matrix, we apply Principal Component Analysis (PCA). PCA transforms the data into a set of orthogonal components that capture the most significant variance. The formula for PCA involves computing the eigenvectors and eigenvalues of the covariance matrix of the data. Let X be the data matrix with n samples and d dimensions:

$$\text{Cov}(X) = \frac{1}{n-1} X^T X$$

The eigenvectors (V) and eigenvalues (Λ) of the covariance matrix are then calculated:

$$\text{Cov}(X)V = V\Lambda$$

The data is then projected onto the top k eigenvectors to obtain the reduced dimensionality representation:

$$X_{\text{PCA}} = XV_k$$

After dimensionality reduction, we use the k-means clustering algorithm to partition the edge servers into two clusters, representing benign and malicious servers. The K-means clustering algorithm aims to partition the data points into k clusters by minimising the within-cluster sum of squares (WCSS). The objective is to minimise the following function:

$$\min_{\{C_1, C_2, \ldots, C_k\}} \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in C_j} \|\mathbf{x}_i - \mu_j\|^2$$

where:

- $\mathbf{x}_i$ represents the data points (e.g., edge servers after PCA dimensionality reduction).

- C_j is the set of data points assigned to cluster j.
- μ_j is the centroid of cluster j, calculated as the mean of the data points in that cluster:

$$\mu_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i$$

- k is the number of clusters (in our, 2 clusters representing benign and malicious servers).
- $\|\mathbf{x}_i - \mu_j\|^2$ is the squared Euclidean distance between data point $\mathbf{x}_i$ and the centroid μ_j.

The goal of K-means is to assign each data point to one of the k clusters such that the sum of squared distances from the points to their respective cluster centroids is minimised. The centroids are updated iteratively by calculating the mean of the points assigned to each cluster. The algorithm alternates between assigning points to the nearest centroid and recalculating the centroids until the assignments no longer change significantly, indicating convergence.

To evaluate the effectiveness of the clustering, we compute the Silhouette score, which measures how similar each point is to its own cluster compared to other clusters. The Silhouette score for a single sample is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where $a(i)$ is the average distance from the i-th point to the other points in the same cluster, and $b(i)$ is the minimum average distance from the i-th point to points in a different cluster. The overall Silhouette score is the mean Silhouette score of all samples:

$$S = \frac{1}{n} \sum_{i=1}^{n} s(i)$$

A high Silhouette score indicates well-separated clusters, suggesting that the model discrepancy scores can effectively differentiate between benign and malicious servers. This clustering analysis not only helps in visualising the attack impact but also provides insights into the validity of model discrepancy scores to detect the malicious servers.

5.6 Experimental Setup

In this section, we design experiments to evaluate the model discrepancy scores during 3LHFL and 4LHFL under different attacks. Figure 5.2 illustrate overview of the experimental method.

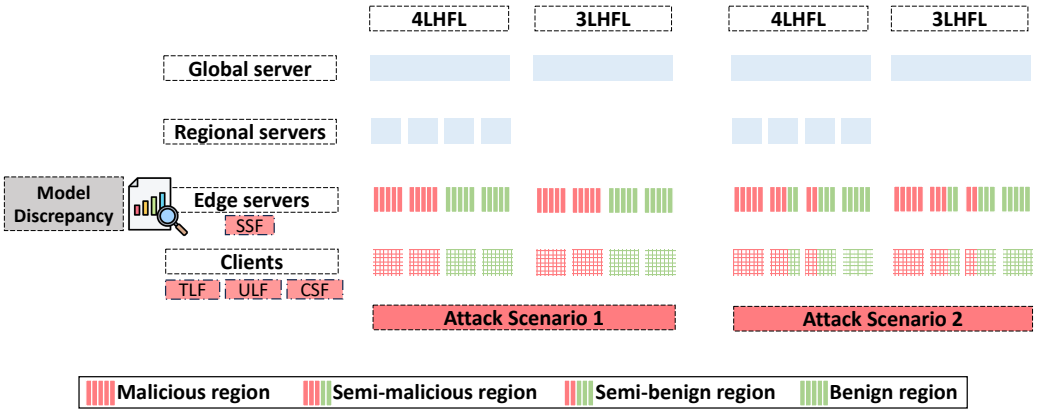

Fig. 5.2 Overview of the Experimental Method. The 3LHFL and 4LHFL architectures were simulated under two attack scenarios: Regional Attack and Distributed Attack. Various attack methods were tested within each scenario, including Targeted Label Flipping (TLF), Untargeted Label Flipping (ULF), Client-Side Sign Flipping (CSF), and Server-Side Sign Flipping (SSF). For both scenarios, 50% of client (or edge servers) are malicious.

5.6.1 Datasets

Three datasets are used to train the models in a HFL setup. **MNIST:** A widely-used dataset of handwritten digit images, consisting of 70,000 images (60,000 for training and 10,000 for testing). Each image is a 28x28 grayscale image representing one of ten digits (0-9). **Fashion-MNIST:** Created by Zalando, this dataset contains 70,000 grayscale images of various fashion items, divided into 60,000 training images and 10,000 test images. Each image is a 28x28 pixel representation of one of ten classes of fashion items. **CIFAR-10:** A dataset of 60,000 color images, each with dimensions of 32x32 pixels, divided into 50,000 training images and 10,000 test images. It includes ten classes of objects such as airplanes, automobiles, birds, cats, and others. The datasets are partitioned with a non-IID degree of 0.01 using the Dirichlet distribution [155]. This specific degree exemplifies a highly non-IID scenario, where each client receives samples from only a small subset of the available classes.

5.6.2 HFL setting

The HFL process involves 100 **clients** that engage in communication with the server for the purpose of image classification model training. Each client trains a local classifier model to classify images. **Global Server:** There is one global server in each learning paradigm at level 0, performing the FedAvg aggregation rule for 20 aggregation rounds. The global server is assumed to be highly secure and uncompromised. **Edge Servers:** There are 20 edge servers

distributed across 4 regions(Regions 1, Region 2,Region 3 and Region 4), each interacting with 5 clients. Edge servers perform the FedAvg aggregation rule for two aggregation rounds. **Regional Servers (4LHFL only):** Regional servers aggregate updates from edge servers in the same region using the FedAvg aggregation rule for three aggregation rounds.

Two different convolutional neural network (CNN) architectures are used for the client's local classifier model. **For MNIST and Fashion-MNIST:** A CNN with two 3x3 convolution layers (first with 32 channels, second with 64, each followed by 2x2 max-pooling), a fully connected layer with 512 units and ReLU activation, and a final softmax output layer with 10 outputs. **For CIFAR-10:** A CNN with two 3x3 convolution layers with 32 channels followed by 2x2 max pooling, another two 3x3 convolution layers with 64 channels followed by 2x2 max pooling, a fully connected layer with 512 units and ReLU activation, and a final softmax output layer with 10 outputs.

Each client employs categorical cross-entropy as the loss function and uses the Adam optimiser. The batch size is set to 32 with 1 epoch for MNIST and Fashion-MNIST, and 64 with 6 epochs for CIFAR-10.

5.6.3 Attack Scenarios

We explore the pairwise model discrepancy scores of 20 edge servers distributed across 4 regions within hierarchical federated learning (HFL) architectures under various attack conditions. To simulate a worst-case scenario and thoroughly assess the system's resilience, 50% of the nodes are set as malicious, compromising 10 edge servers and 50 clients. This proportion is chosen to ensure that the adversarial influence is significant enough to observe the impact of attacks clearly across different discrepancy analyses. An edge server is classified as malicious if it is either directly compromised or connected to compromised clients. The severity of maliciousness in a region is determined by the proportion of malicious servers present. Two distinct attack scenarios are implemented: the Regional Attack and the Distributed Attack. Future work will include treating the proportion of malicious nodes as a hyperparameter and conducting an ablation study to explore varying attack intensities.

In the Regional Attack scenario, Region 1 and 2 are completely malicious, meaning all edge servers within these regions are compromised. Meanwhile, Region 3 and 4 remain entirely benign. This setup simulates a situation where large geographical areas or organisational domains are fully compromised, offering insights into the impact of highly concentrated malicious activity on the federated learning process.

In contrast, the Distributed Attack scenario presents a more nuanced landscape. Here, Region 1 is completely malicious, while Region 2 is partially compromised, containing a mix of malicious and benign edge servers with malicious servers are more the benign

ones. Region 3 is predominantly benign with a small fraction of malicious servers, and Region 4 remains entirely benign. Thus, Regions 1, Region 2,Region 3 and Region 4 are called malicious Regions, Semi-malicious Region,Semi-benign Region and Benign Region, respectively. This scenario mimics a more realistic and complex attack environment, where malicious entities are dispersed within generally benign regions. The varying levels of regional compromise pose significant challenges for detecting and mitigating attacks in federated learning systems.

To evaluate the impact of these attack scenarios, we created four experimental configurations within hierarchical federated learning architectures: 3LHFL1, 3LHFL2, 4LHFL1, and 4LHFL2. The 3LHFL1 configuration involves a 3-Level Hierarchical Federated Learning system subjected to the Regional Attack conditions, while 3LHFL2 applies the Distributed Attack conditions to the same 3-level architecture. Similarly, 4LHFL1 and 4LHFL2 apply the Regional and Distributed Attack conditions, respectively, to a 4-Level Hierarchical Federated Learning system.

These experimental setups allow for a detailed evaluation of HFL dynamics under different attack conditions by analyzing model discrepancy scores. Our approach provides insights into how the distribution of malicious servers within regions affects the model discrepancy, contributing to the development of more robust and secure federated learning frameworks capable of withstanding diverse and sophisticated attack vectors.

5.7 Results

5.7.1 Temporal Analysis of Edge Server Model Discrepancy scores

In this section, we analyze the temporal evolution of pairwise model discrepancy scores between edge servers across multiple aggregation rounds. The analysis focuses on how these scores change over time, both under clean training conditions and when subjected to various attack scenarios.

Figure 5.3 visualises the dynamics of the 3LHFL and 4LHFL architectures, highlighting the impact of attacks on model consistency. It contrasts the behaviour of the architectures during normal operation versus under attack, providing insights into how malicious activity disrupts the federated learning process and alters model discrepancy patterns.

Clean training

The analysis revealed distinct patterns in model discrepancy scores under clean and attack conditions for both the 3LHFL and 4LHFL architectures. Under clean conditions, where no

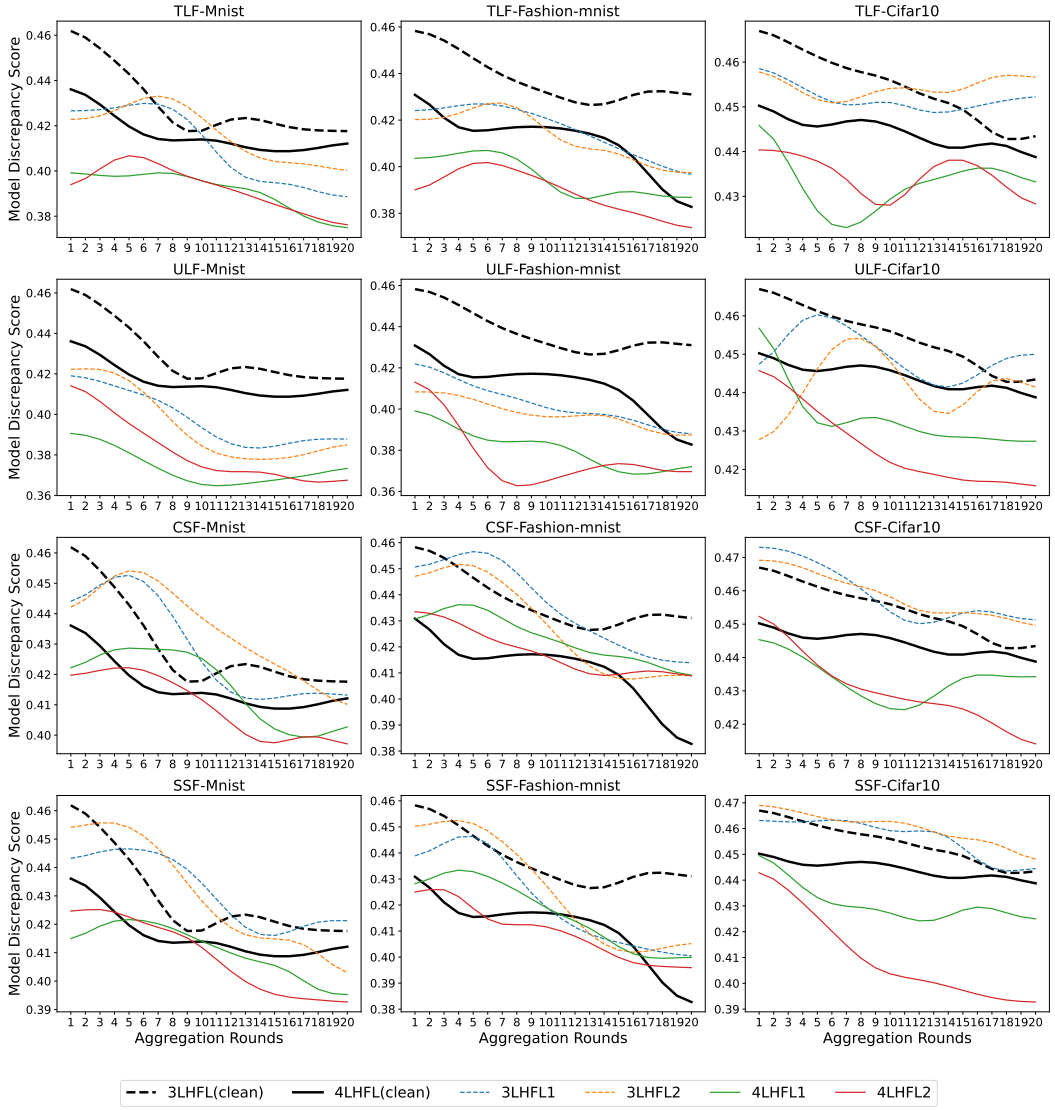

Fig. 5.3 Temporal Analysis of Edge Server Model Discrepancy. The figure illustrates the temporal dynamics of the 3LHFL and 4LHFL architectures during training, comparing the behaviour under clean conditions with that under attack scenarios. The visualisation highlights how attacks impact model consistency over multiple aggregation rounds, revealing key differences in model discrepancy patterns between benign and malicious conditions.

attacks were present, the model discrepancy was initially high due to the non-IID nature of the data across different clients. As training progressed, the aggregation process effectively reduced this discrepancy, leading to a more aligned and coherent global model. Both the 3LHFL and 4LHFL architectures exhibited a steady decrease in model discrepancy score,

with the 4LHFL architecture demonstrating slightly lower initial discrepancy scores due to its additional hierarchical level, which aids in better smoothing out variations.

Regional attack scenario

Under attack conditions, different attack scenarios impacted the model discrepancy scores in various ways. In the regional attack scenario, the 3LHFL architecture displayed higher initial discrepancies due to the influence of entirely malicious regions. These discrepancy scores fluctuated throughout the aggregation rounds as the malicious updates impacted the model, but overall, the discrepancies under attack conditions remained lower than those in clean conditions due to the alignment of malicious updates towards a corrupted state. These alignments create the appearance of reduced discrepancies among model updates, but they actually steer the global model towards a maliciously biased or corrupted state. The 4LHFL architecture exhibited a similar pattern but with lower discrepancies compared to 3LHFL. The additional hierarchical level in 4LHFL smooths out the discrepancy scores more effectively.

Distributed attack scenario

In the distributed attack scenario, the 3LHFL architecture showed an initial high discrepancy scores due to the presence of mixed malicious activities across regions. This discrepancy decreased steadily as the model progressed through the aggregation rounds but remained lower than the clean condition due to the same alignment of malicious updates towards a corrupted state. The 4LHFL architecture, with its additional hierarchical level, demonstrated lower discrepancy scores under distributed attack conditions compared to 3LHFL. The discrepancy scores were higher initially but showed a smoother decline towards the later aggregation rounds.

The detailed discrepancy analysis reveals that while both 3LHFL and 4LHFL architectures exhibit lower model discrepancy scores under attack conditions than in clean conditions, The additional hierarchical level in 4LHFL helps in distributing the updates more evenly and reduces the immediate impact of any single malicious update. As a result, the overall model updates appear more coherent and less discrepant. However, the observed lower discrepancy scores under attack conditions do not indicate improved performance but rather reflect the alignment of malicious updates towards a corrupted state. These alignments create the appearance of reduced discrepancy scores among model updates, but they actually steer the global model towards a maliciously biased or corrupted state.

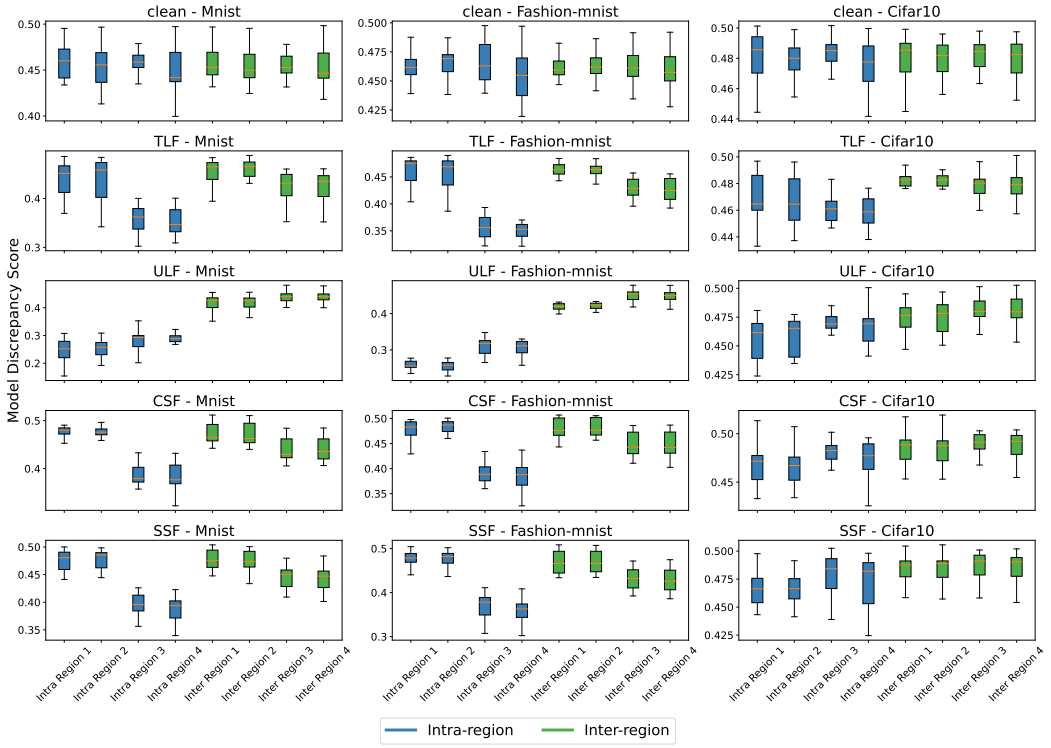

Fig. 5.4 Spatial Analysis of Model Discrepancy Scores during Architecture Under clean condition and Regional Attack (3LHFL1).

5.7.2 Spatial Analysis of Edge Server Model Discrepancy Scores

This section presents the spatial analysis of edge server model discrepancy scores by examining and comparing the distributions of pairwise model discrepancies within the same region (intra-region) and across different regions (inter-region) during the first aggregation round.

Boxplots are used to analyse the median and distribution (consistency) of these discrepancy scores for both clean and attack conditions. The analysis is broken down by region type, comparing malicious, semi-malicious, semi-benign and benign regions. Figure 5.4 to 5.7 illustrate these comparisons, revealing how model discrepancies behave in different scenarios. Through these visualisations, we gain insights into how attacks alter model consistency across regions, and how regional classification (malicious, semi-malicious, or benign) affects the distribution and median of discrepancy scores under different learning conditions.

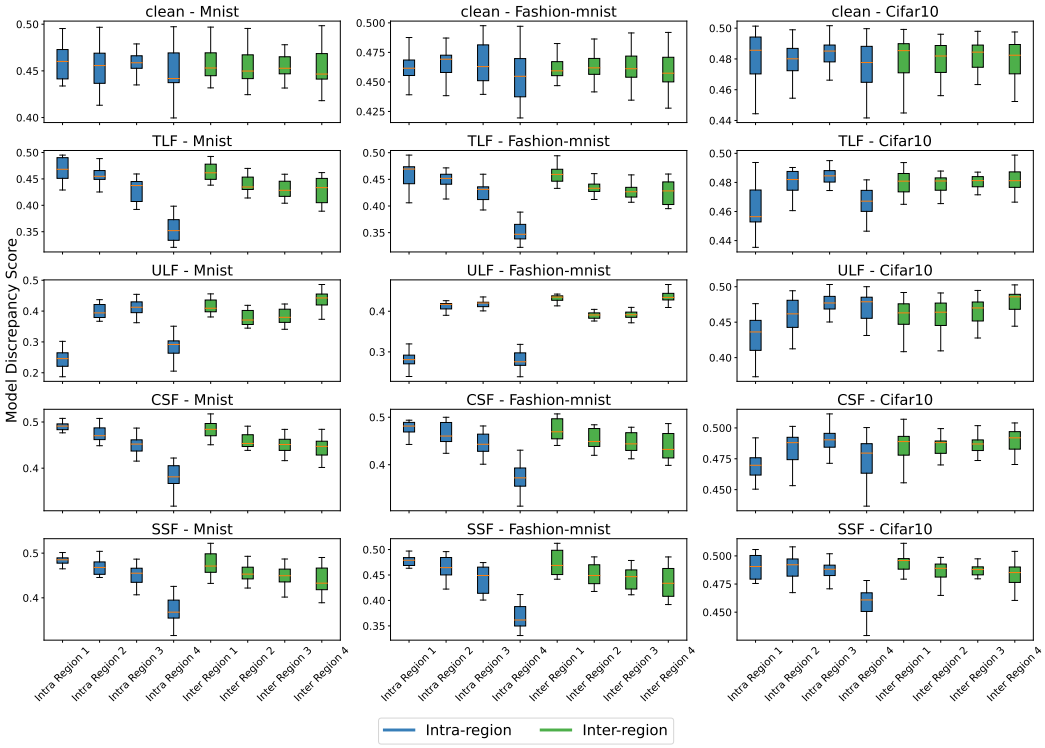

Fig. 5.5 Spatial Analysis of Model Discrepancy Scores during Architecture Under clean condition and Distributed Attack (3LHFL2).

3LHFL architecture

In the 3LHFL architecture, the clean learning scenario reveals that the median intra-region and inter-region discrepancy scores are nearly identical, indicating uniform model discrepancies across edge servers in the absence of attacks.

However, under attack conditions, the results diverge significantly. In the regional attack scenario (3LHFL1), malicious regions show substantially higher intra-region discrepancy scores than benign regions, particularly under Targeted Label Flipping (TLF), Client-Side Sign Flipping (CSF), and Server-Side Sign Flipping (SSF) attacks. This suggests greater variability and instability in model updates within malicious regions. Interestingly, for Untargeted Label Flipping (ULF), discrepancy scores within malicious regions are lower than those within benign regions. Inter-region discrepancy scores show a similar pattern but are higher overall, reflecting the wider impact of regional attacks across the network.

In the distributed attack scenario (3LHFL2), malicious regions again exhibit the highest intra-region discrepancy scores under TLF, CSF, and SSF attacks, followed by semi-malicious, semi-benign, and benign regions. However, ULF attacks result in lower discrepancy scores

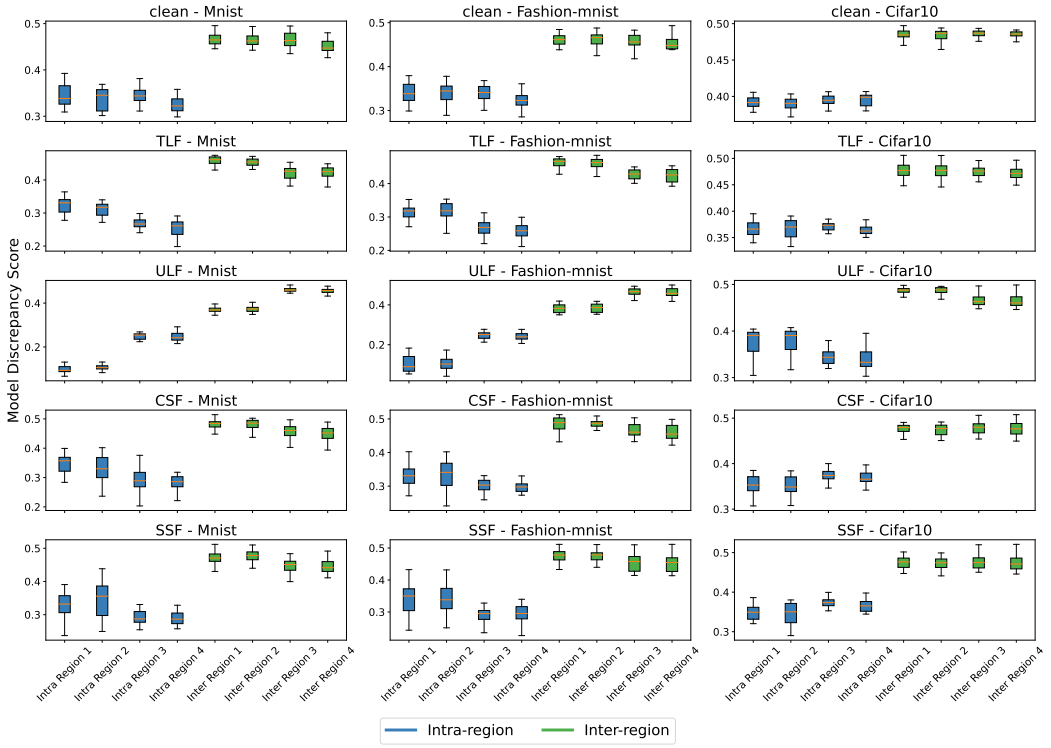

Fig. 5.6 Spatial Analysis of Model Discrepancy Scores during Architecture Under clean condition and Regional Attack (4LHFL1).

within malicious regions, with benign regions following. Semi-malicious and semi-benign regions display higher and comparable discrepancy scores. Inter-region discrepancies mirror these intra-region trends, with ULF attacks yielding higher discrepancies between benign and malicious regions.

4LHFL architecture

The clean learning scenario in the 4LHFL architecture presents a different pattern compared to the 3LHFL architecture. Here, inter-region discrepancy scores are higher than intra-region discrepancies but remain relatively uniform across regions, suggesting consistent model updates in the absence of attacks.

In the regional attack scenario (4LHFL1), similar to 3LHFL1, malicious regions exhibit higher discrepancy scores than benign regions, except in the case of ULF, which shows the opposite trend. However, the differences in discrepancy scores between malicious and benign regions are less pronounced in the 4LHFL architecture compared to the 3LHFL architecture,

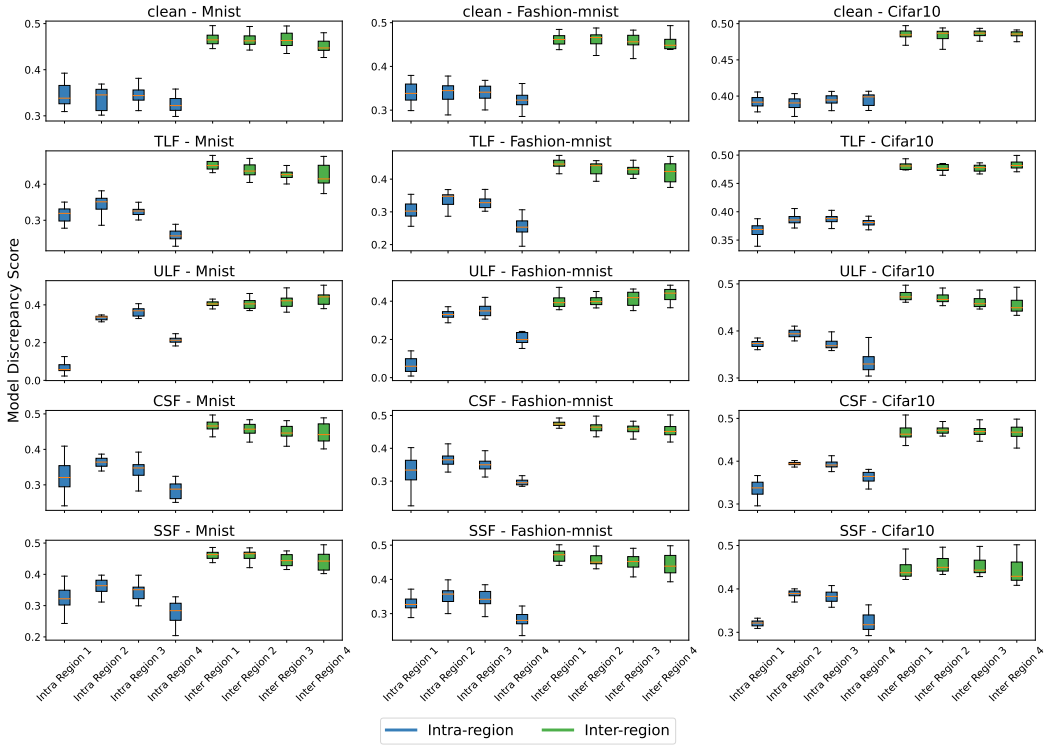

Fig. 5.7 Spatial Analysis of Model Discrepancy Scores during Architecture Under clean condition and Distributed Attack (4LHFL2).

indicating more consistency. Discrepancy scores for benign and malicious regions are closer and more stable than in the 3LHFL architecture.

In the distributed attack scenario (4LHFL2), the differences in discrepancy scores between malicious, semi-malicious, semi-benign, and benign regions are smaller than in the 3LHFL architecture. All attacks show the lowest discrepancy scores within malicious regions, followed by benign regions, while semi-malicious and semi-benign regions display higher and similar discrepancy scores.

The results demonstrate that model discrepancy scores during hierarchical federated learning vary significantly under different attack scenarios. The 3LHFL architecture is more sensitive to both regional and distributed attacks, with higher discrepancy scores within and between regions, especially under TLF, CSF, and SSF attacks. In contrast, the 4LHFL architecture exhibits smaller discrepancies and more consistent scores across regions.

5.7.3 Clustering Analysis Based on Edge Server Model Discrepancy Scores

This section presents the clustering analysis of edge server model discrepancy scores, aimed at evaluating the effectiveness of these scores in distinguishing between benign and malicious servers across various attack scenarios in both 3LHFL and 4LHFL architectures. To achieve this, we apply Principal Component Analysis (PCA) for dimensionality reduction, followed by k-means clustering to partition the edge servers based on their model discrepancy scores. The quality of the clustering is assessed using Silhouette scores.

Figures 5.8 to 5.11 depict the PCA components of model discrepancy scores and the resulting k-means clustering for edge servers. Each figure includes Silhouette scores to reflect the clustering quality. In these visualisations, each point corresponds to an edge server's model discrepancy score with respect to other servers, projected in PCA-reduced dimensions. Edge servers are labelled as 'm' for malicious and 'b' for benign, providing a clear distinction in the clustering results based on their behaviour under attack conditions.

3LHFL architecture

Under clean learning conditions, the 3LHFL architecture demonstrates relatively low clustering quality, with Silhouette scores ranging from 0.36 to 0.55. These scores establish a baseline for the clustering performance in the absence of any malicious activity.

However, under attack conditions, the 3LHFL architecture exhibits significantly higher clustering quality. In the Regional Attack scenario (3LHFL1), the first principal component effectively separates edge server model discrepancy scores into two distinct clusters: benign servers and malicious servers. The clustering quality is consistently high across all datasets, though CIFAR-10 shows slightly lower scores, indicating a less pronounced differentiation between malicious and benign servers. Silhouette scores range from 0.77 to 0.93, with CSF and SSF attacks yielding better clustering quality for CIFAR-10 than TLF and ULF attacks. In contrast, for the MNIST and Fashion MNIST datasets, benign servers within benign regions exhibit high intra-cluster cohesion, with similar model discrepancy patterns leading to tight clustering in PCA space. This pattern is less consistent for CIFAR-10, where more variability is observed.

In the Distributed Attack scenario (3LHFL2), clustering quality remains strong, with Silhouette scores ranging from 0.77 to 0.93. A noticeable gradation in clustering quality is seen, particularly for the CIFAR-10 dataset. As with 3LHFL1, the first principal component effectively separates discrepancy scores into two clusters: benign and malicious servers. High intra-cluster cohesion is again observed among benign servers, even when they are

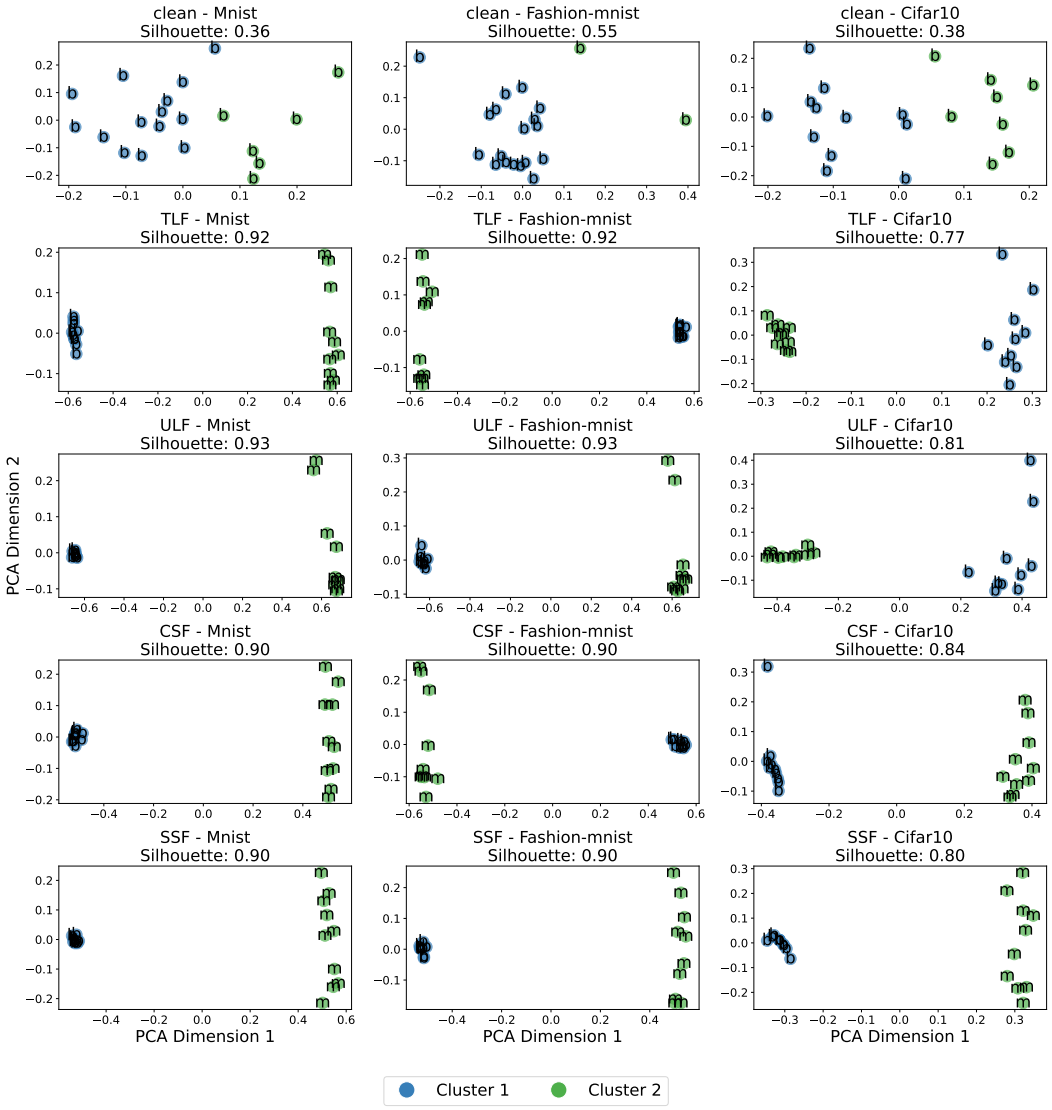

Fig. 5.8 Clustering Analysis of Edge Server Model Discrepancy Scores in 3LHFL Architecture Under Regional Attack(3LHFL1).

located in semi-malicious or semi-benign regions, as there are no distinct regional boundaries in this scenario.

The high Silhouette scores in both the 3LHFL1 and 3LHFL2 architectures indicate a strong distinction between benign and malicious servers under attack conditions. These results suggest that the attacks—Regional Attack for 3LHFL1 and Distributed Attack for 3LHFL2—introduce significant variability in the model discrepancy scores of malicious servers relative to benign ones, making them easily separable. These high Silhouette scores

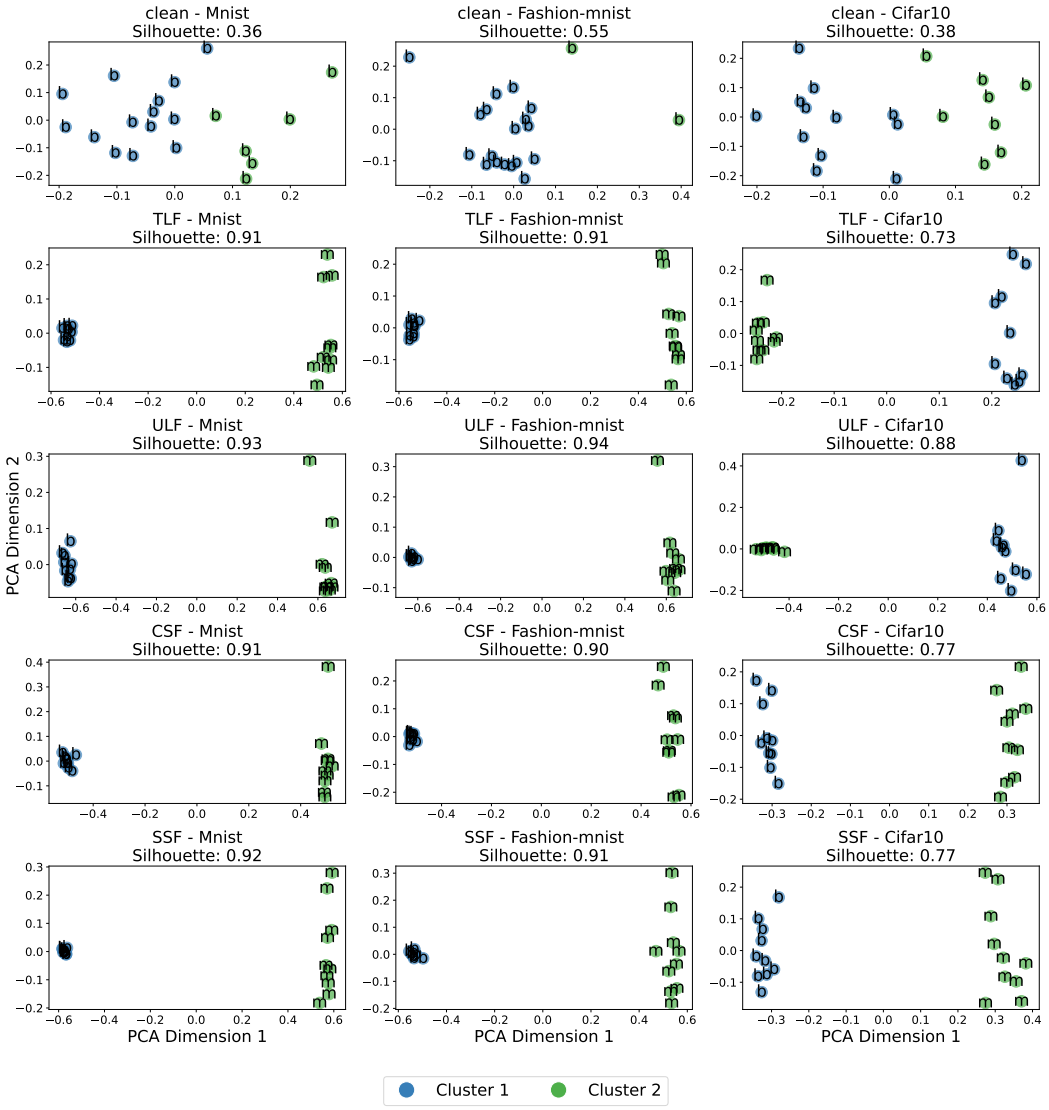

Fig. 5.9 Clustering Analysis of Edge Server Model Discrepancy Scores in 3LHFL Architecture Under Distributed Attack(3LHFL2).

indicate that the model discrepancy scores are robust enough to clearly distinguish between benign and malicious servers.

4LHFL architecture

Under clean learning conditions, the 4LHFL architecture demonstrates higher clustering quality across all datasets compared to the 3LHFL architecture, with Silhouette scores

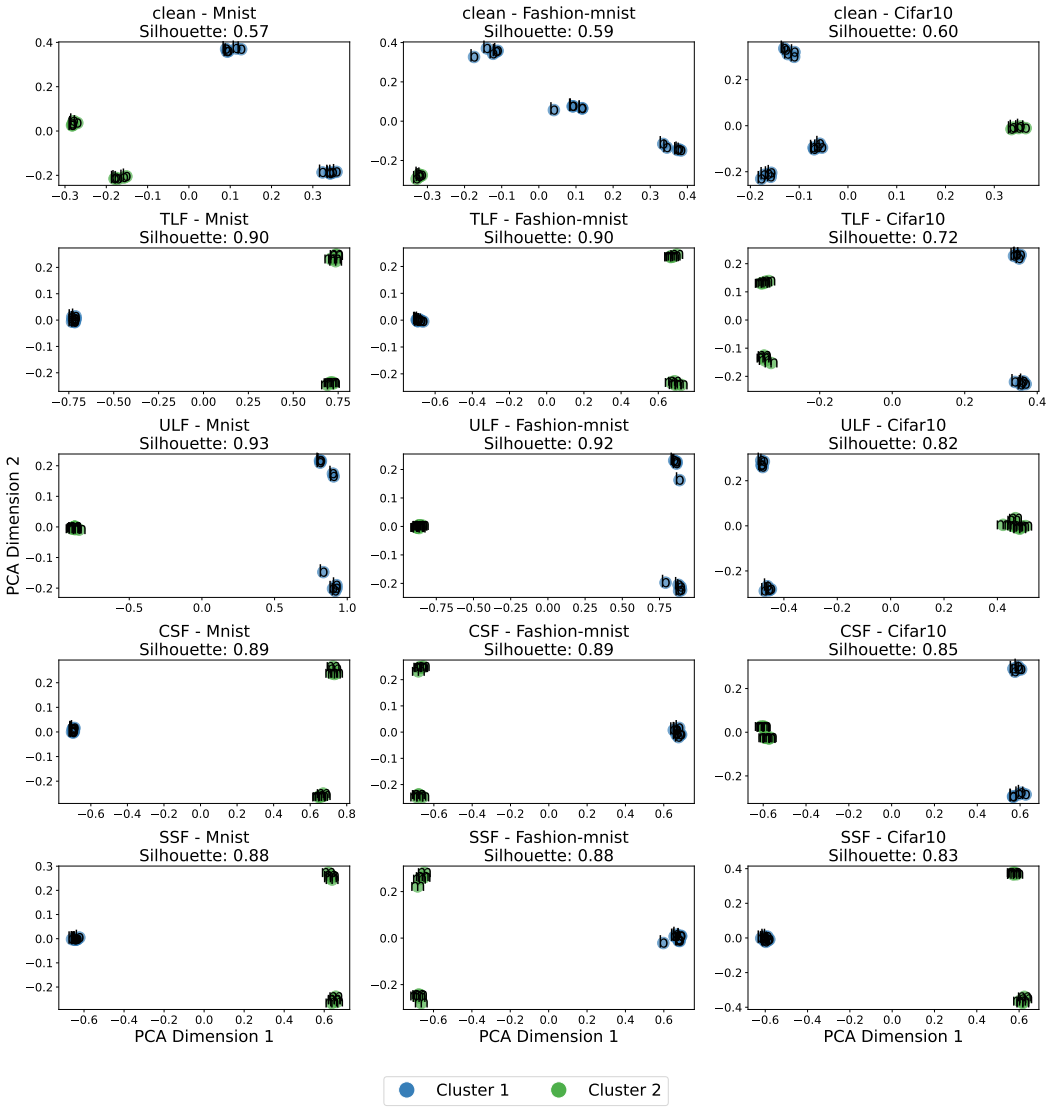

Fig. 5.10 Clustering Analysis of Edge Server Model Discrepancy Scores in 4LHFL Architecture Under Regional Attack(4LHFL1).

ranging from 0.57 to 0.60. These scores provide a baseline for the clustering performance in the absence of any malicious activity.

Under Regional Attack conditions (4LHFL1), the 4LHFL architecture exhibits high clustering quality, similar to the 3LHFL architecture, with Silhouette scores ranging from 0.72 to 0.93. CSF and SSF attacks result in more clearly defined clusters than TLF and ULF attacks for the CIFAR-10 dataset, though this trend does not hold for other datasets. Notably, both benign and malicious servers across all datasets and attack types show high intra-cluster

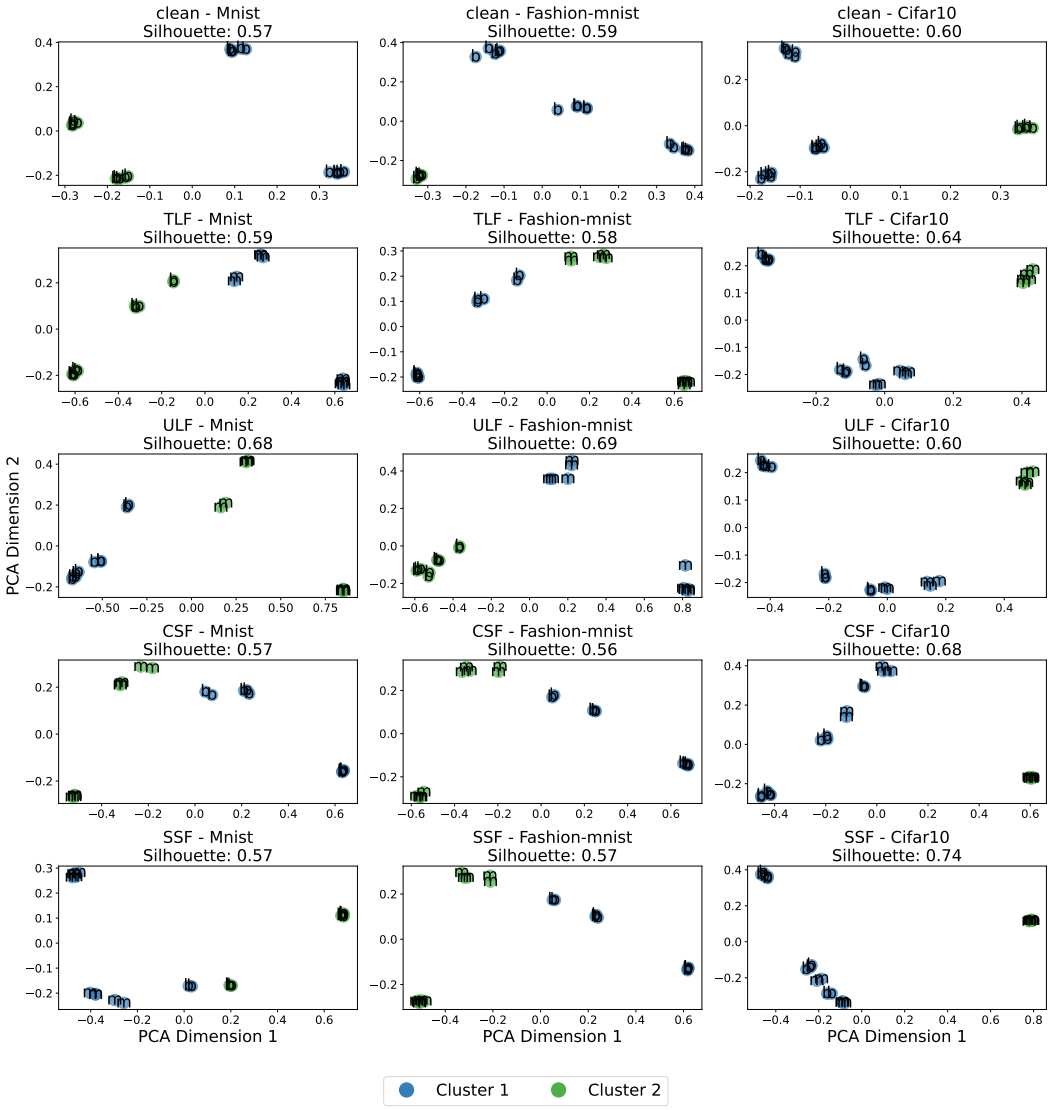

Fig. 5.11 Clustering Analysis of Edge Server Model Discrepancy Scores in 4LHFL Architecture Under Distributed Attack(4LHFL2).

cohesion. Interestingly, the malicious cluster can often be further divided into two sub-clusters, reflecting that edge servers within the same malicious region display similar model discrepancy patterns, while those in other malicious regions exhibit distinct behaviours. This pattern is particularly evident for TLF, CSF, and SSF attacks. However, in the case of ULF attacks, the benign cluster sometimes splits into two sub-clusters, while the malicious cluster remains tightly cohesive. The high Silhouette scores under Regional Attack conditions in the 4LHFL architecture indicate a strong ability to maintain clear separation between benign and

malicious servers. This suggests that the architecture, through its model discrepancy scores, is robust against regional attacks, effectively distinguishing between benign and malicious behaviours.

Under Distributed Attack conditions (4LHFL2), the 4LHFL architecture exhibits the lowest clustering quality compared to 3LHFL1, 3LHFL2, and 4LHFL1, with Silhouette scores ranging from 0.56 to 0.74. These scores are closer to the clean learning baseline, indicating a diminished ability to clearly differentiate between benign and malicious servers. Although the clustering still effectively separates most benign and malicious servers, the quality decreases in semi-benign and semi-malicious regions, where some malicious servers are clustered together with benign ones. This decline in clustering quality is attributed to the growing similarity in model discrepancy patterns between malicious and benign servers in these mixed regions, resulting in closely positioned points in the PCA space. This indicates that the distributed attack strategy, along with the additional aggregation layers, blurs the distinction between malicious and benign behaviours.

5.8 Disscusion

5.8.1 Key Factors Influencing Model Discrepancy in HFL

Several factors influence model discrepancy in hierarchical federated learning (HFL) systems, impacting the system's ability to detect and mitigate malicious activities. These factors shape how effectively model discrepancies reflect the presence of attacks, and ultimately influence the dynamics of the federated learning process.

- Data Heterogeneity:

  In HFL systems, data is typically non-IID (non-independent and identically distributed) across different clients. This data heterogeneity arises because clients often have unique data distributions due to varying local environments or user behaviours. When data is highly heterogeneous, local model updates from different clients tend to diverge naturally, leading to increased model discrepancies even in benign conditions. This can complicate the detection of malicious activity, as the variability caused by natural data distribution differences may mask discrepancies introduced by attacks. Consequently, distinguishing between normal variations and malicious discrepancies becomes more challenging in the presence of high data heterogeneity.

- Attack Sophistication:

The level of sophistication of attacks plays a critical role in influencing model discrepancy scores. Simple attacks may cause clear and noticeable discrepancies, making them easier to detect. However, more sophisticated attacks, such as targeted data poisoning, model poisoning, or adaptive adversarial attacks, can be designed to mimic benign behaviours closely. These attacks can reduce the observable differences between malicious and benign servers, thus lowering model discrepancy scores. The adaptability of these advanced attacks makes them more challenging to detect, as they are capable of evading traditional discrepancy-based detection methods by generating subtle discrepancies that resemble natural variations in model updates.

- Aggregation Layers:

  HFL systems involve a hierarchical structure with multiple aggregation layers, such as edge, regional, and global aggregations. Each of these layers combines model updates from the lower levels to create a consolidated update for the next level. This multi-level aggregation process adds complexity to the management of model discrepancies. Intermediate aggregations may obscure discrepancies introduced by malicious servers, especially if benign and malicious updates are combined before reaching the higher aggregation layers. As a result, the impact of an attack may become diluted or more difficult to trace back to the malicious source. Properly designing aggregation methods that account for these complexities is crucial for enhancing the robustness of the system.

- Metrics Used for Discrepancy Measurement:

  The choice of metrics used to measure model discrepancies has a direct impact on the system's ability to detect malicious activities. Common metrics include dissimilarity, distance, uncorrelation, and divergence. Each of these metrics captures different aspects of the discrepancies between model updates. The integration of these metrics into a unified discrepancy score improves the robustness of the detection system by offering a more comprehensive analysis of model behaviours. Different attacks may affect different aspects of the model, and using multiple metrics allows for a broader and more accurate assessment of potential threats.

5.8.2 Challenges of Controlled Simulations in Real-World Federated Learning

This study evaluates attack scenarios and model discrepancy detection in a controlled simulation environment, which allows precise manipulation of variables, controlled deployment

of attacks, and repeatable experiments. While this approach is essential for systematically analyzing the effects of different adversarial strategies on Hierarchical Federated Learning (HFL), it introduces certain limitations when translating findings to real-world conditions.

One key difference is the predictability of attack behaviors in a controlled setting. In real-world federated learning systems, adversaries may employ adaptive strategies, continuously refining their attacks based on model updates and detection mechanisms. In contrast, the attacks tested in this study are static, meaning malicious nodes do not alter their strategies over training rounds. Additionally, real-world adversaries may combine multiple attack types or employ evasion techniques, such as obfuscating malicious updates to bypass discrepancy detection.

Another limitation is the assumption of ideal network conditions. The simulation assumes timely model updates, stable communication channels, and consistent client participation. However, real-world federated learning systems operate in dynamic environments where network delays, packet loss, and device failures may impact aggregation and detection performance. These factors can influence the effectiveness of discrepancy-based attack detection, as missing updates or delayed contributions may mimic or mask adversarial behaviors.

The scale and diversity of data distributions also differ. In a controlled setting, data heterogeneity is defined using a Dirichlet-based non-IID partitioning. While this reflects statistical heterogeneity, real-world datasets may exhibit temporal shifts, evolving patterns, and unexpected biases that challenge the generalizability of fixed experimental setups. Additionally, client participation in real-world HFL is often unpredictable, with clients joining and leaving dynamically, whereas the experiments assume a fixed number of participants.

Implications for Generalizability While the controlled environment enables rigorous evaluation, applying the findings to real-world federated learning deployments requires caution. The effectiveness of the Model Discrepancy Score and multi-metric analysis may vary when facing adaptive adversaries, unreliable communication, or shifting data distributions. Future work will focus on deploying the approach in practical federated learning systems, incorporating real-world network constraints, and testing adaptive adversarial strategies to enhance generalizability.

5.8.3 Scope and Limitations of Attack Scenarios Tested

This study evaluates the impact of adversarial attacks on Hierarchical Federated Learning (HFL) through Regional and Distributed attack scenarios. These scenarios simulate environments where malicious nodes either concentrate in specific regions or are spread across multiple regions. The attacks tested include Targeted Label Flipping (TLF), Untargeted Label

Flipping (ULF), Client-Side Sign Flipping (CSF), and Server-Side Sign Flipping (SSF), all of which focus on manipulating model updates without requiring knowledge of the global model. The study provides insights into worst-case scenarios by assuming a static attack strategy, where malicious nodes do not adapt their behavior over time, and by setting 50% of nodes as malicious, ensuring significant adversarial influence on model aggregation.

Despite these contributions, the attack scenarios have certain limitations. The tested attacks are non-adaptive, meaning malicious nodes do not change their strategies across training rounds, whereas real-world adversaries could adapt to evade detection. Additionally, the study does not consider gradient-based attacks, such as Byzantine-resilient model poisoning or gradient inversion attacks, where adversaries actively craft malicious updates based on model parameters. Collusion attacks, where multiple malicious nodes collaborate to manipulate updates in a way that remains undetectable by the aggregation mechanism, are also not included. Furthermore, communication-based attacks, such as delaying or selectively dropping updates to mislead the global model, are not part of the scope.

Future work will extend this research by incorporating adaptive attack strategies and gradient-based manipulations to evaluate how dynamically evolving adversarial behaviors impact model discrepancy. Additionally, exploring collusion-resistant aggregation methods and assessing the resilience of HFL under real-world attack variations will further enhance the robustness of federated learning security.

5.9 Summary

This study investigates the complexities and vulnerabilities of Hierarchical Federated Learning (HFL) systems under various attack scenarios, emphasising the critical need for robust discrepancy management to ensure security and effectiveness in distributed environments. We introduced a comprehensive Model Discrepancy score, integrating multiple metrics—Dissimilarity, Distance, Uncorrelation, and Divergence—to effectively measure discrepancies in model updates.

Our experimental analysis of two prominent HFL architectures (3-level HFL and 4-level HFL) under conditions of high data heterogeneity reveals several key insights. First, the Model Discrepancy score proves to be a reliable tool for detecting malicious servers and effectively distinguishing between benign and malicious servers. Second, spatial and clustering analyses highlight how attacks impact the dynamic behaviour of HFL systems. While benign servers exhibit high intra-cluster cohesion and maintain clear differentiation from malicious servers, the 4LHFL architecture faces challenges under distributed attack

conditions, particularly when the discrepancy patterns of benign and malicious servers converge.

These findings underscore the importance of a multifaceted approach to discrepancy management, incorporating various metrics to enhance the detection and mitigation of malicious activities in HFL systems. Additionally, the study identifies key factors influencing model discrepancy, such as data heterogeneity, attack sophistication, aggregation layers, and the choice of discrepancy metrics. Understanding how these factors interact with the dynamic behaviour of HFL systems under attack is crucial for developing effective and robust aggregation techniques.

Future research should extend the analysis to more diverse datasets and federated learning architectures, integrate advanced anomaly detection techniques, and develop robust aggregation rules tailored to the dynamic nature of HFL systems under attack. By addressing these areas, the robustness and resilience of HFL systems can be significantly enhanced, ensuring their secure and reliable deployment in real-world applications.

Chapter 6

Conclusions and Future Work

6.1 Summary of Key Findings

In this thesis, we provide insights into effective defense mechanisms and comprehensive security assessments aimed at enhancing the security of Distributed Deep Learning (DL) systems. Through addressing the key research questions, this work offers practical solutions and assessments for safeguarding both the availability and integrity of these systems within the client-edge-cloud continuum. Below, we summarize key findings by answering each research question directly.

1. How can early-stage IoT botnets be effectively detected at the edge to prevent DDoS attacks in data-centric distributed DL systems?

The research introduces an edge-based detection system (EDIT) that leverages a Modular Neural Network (MNN) to detect early-stage IoT botnet activities at Multi-access Edge Computing (MEC) servers. By identifying abnormal traffic patterns during initial botnet phases (e.g., scanning and command-and-control), EDIT prevents large-scale Distributed Denial of Service (DDoS) attacks before they can cause system disruption. The modular design of the neural network supports parallel processing, enabling real-time detection in large-scale environments such as smart cities. This parallelism not only speeds up detection but also reduces false negatives, ensuring system availability is maintained in real-time operations.

2. How does the HFL architecture impact the robustness of HFL systems against adversarial attacks, and what defense mechanisms can enhance their security?

The study finds that while Hierarchical Federated Learning (HFL) enhances scalability and training efficiency, its hierarchical structure also introduces specific vulnerabilities. Targeted attacks, such as backdoor attacks, pose a significant risk to HFL systems, particularly in multi-level models (e.g., 4-level HFL). To counteract these threats, the research advocates

for adversarial training (AT) and Neural Cleanse (NC) as effective defense mechanisms. AT helps reduce misclassifications by training models on adversarial inputs, while NC mitigates backdoor attacks by detecting and cleaning compromised neurons in the model. The study further reveals that increased hierarchy does not automatically translate into improved security, as 4-level HFL systems were found to be more susceptible to backdoor attacks than simpler, 2-level federated learning (FL) models.

3. How do different attack scenarios impact the dynamics of Hierarchical Federated Learning (HFL) systems, and what factors influence model discrepancies during these attacks?

The dynamics of HFL systems under attack are significantly influenced by the level of hierarchy and data heterogeneity across nodes. A novel Model Discrepancy score is introduced to measure inconsistencies in model updates and to detect malicious activities. The research shows that in high-data heterogeneity environments, discrepancies can emerge even among benign nodes, complicating the identification of attacks. In more hierarchical models (e.g., 4-level HFL), multi-level aggregation further complicates the detection of malicious updates, as overlapping discrepancy patterns between benign and malicious nodes obscure attack traces. This highlights the need for advanced aggregation methods that can account for such complexities. Factors such as data heterogeneity, attack sophistication, and the number of aggregation layers play key roles in influencing model discrepancies, making it essential to design robust aggregation techniques to ensure system integrity.

The three chapters in this thesis collectively enhance the security of Distributed Deep Learning (DL) across the client-edge-cloud continuum, each addressing a specific challenge in securing these architectures. Chapter 3 focuses on availability threats in data-centric Distributed DL applications, where detecting early-stage botnet activities at the edge using a Modular Neural Network (MNN) prevents DDoS attacks that could disrupt communication and block updates. Meanwhile, Chapters 4 and 5 focus on the integrity of Hierarchical Federated Learning (HFL), a key architecture in privacy-centric Distributed DL applications, by assessing its resilience against adversarial attacks and introducing the Model Discrepancy Score (MDS) to detect hidden manipulations. Addressing the security of both data-centric and privacy-centric DL architectures provides valuable insight into the need to consider different security aspects based on application requirements. Furthermore, the botnet detection approach in Chapter 3 can also protect HFL training (Chapters 4 and 5) by ensuring uninterrupted communication at the edge, preventing attackers from using IoT botnets to disrupt federated learning processes. Conversely, the integrity-focused techniques from Chapters 4 and 5 can help secure the model availability in Chapter 3, as both approaches consider Distributed DL systems deployed across the client-edge-cloud continuum. This

interconnected security perspective highlights the importance of multi-layered defenses tailored to the specific challenges of different Distributed DL architectures.

6.2 Limitations

While the research presented in this dissertation has made significant advancements, several limitations should be acknowledged. The proposed edge-based detection system for early-stage IoT botnet detection, though effective, was primarily validated in controlled, simulated environments using specific datasets. The system relies on supervised learning, which necessitates labelled data for training. Although the proof-of-concept results were promising, real-world IoT environments are far more complex. Variables such as network variability, device heterogeneity, and the continuous evolution of threat landscapes introduce challenges that were not fully captured in this study. Furthermore, the reliance on labelled data makes the system less adaptable to new or unseen botnet behaviours that were not present during training. Thus, additional validation is required to ensure that the system can generalise effectively to more diverse, dynamic IoT ecosystems.

In the context of Hierarchical Federated Learning (HFL), the security assessment conducted focused on specific adversarial scenarios like data and model poisoning. While the framework examined particular threats, it does not cover the full spectrum of potential attack vectors that could affect HFL systems. Complex, multi-stage attacks or a combination of various adversarial strategies were not extensively examined. Moreover, the study concentrated on static attack scenarios, which may not accurately reflect the behavior of adversaries in dynamic, real-time environments. As adversaries continuously evolve their tactics, new challenges will likely arise that were not fully addressed in this work.

The Model Discrepancy score, introduced to detect discrepancies in model updates during HFL under attack, also has certain limitations. It operates based on specific assumptions about the nature of malicious behaviour and model updates, which worked well in the controlled settings of this study. However, attackers may employ more subtle and sophisticated techniques, such as gradual manipulations that evade detection. Additionally, the score's performance has primarily been validated in simulated environments, and its effectiveness in more complex, real-world Distributed DL systems—where attack patterns shift and operational conditions vary—requires further investigation. More comprehensive testing is necessary to determine how well the discrepancy score functions in dynamic, less predictable environments where data distributions and adversarial strategies evolve over time.

6.3 Future Work

The limitations identified in this thesis open several avenues for future work.

Edge-Based Botnet Detection: Future work related to the edge-based botnet detection system (EDIT) will involve deploying and evaluating the system in real-world IoT environments. While the system has shown effectiveness in simulated settings, testing it under more complex, dynamic conditions is crucial. This includes addressing challenges such as network variability, device heterogeneity, and evolving botnet threats that arise in practical IoT ecosystems. Further research could focus on enhancing the system's adaptability by incorporating semi-supervised or unsupervised learning approaches, allowing it to detect previously unseen or novel botnet behaviours. Another promising direction would be to optimise the detection system for large-scale IoT deployments, ensuring it can maintain high performance even under conditions with significant data volume and network traffic.

Additionally, real-world deployment should consider computational constraints at the edge, necessitating efficient deep learning models that balance accuracy with low latency. Integrating Explainable AI (XAI) techniques into EDIT could also improve its trustworthiness by providing human-readable explanations for botnet detection decisions, making it more viable for deployment in critical infrastructures such as smart cities and industrial IoT.

Security Assessment of HFL: For the security assessment of Hierarchical Federated Learning (HFL), future research could explore more advanced techniques to strengthen the defense against sophisticated adversarial attacks, particularly those that adapt over time. Enhancing the robustness of HFL against targeted attacks like backdoor attacks remains a critical area of investigation. Additionally, improving defense mechanisms for more complex and diverse real-world data environments is necessary. Future work may involve investigating new aggregation methods that can mitigate subtle adversarial manipulations, especially in highly heterogeneous environments. Another key area of research could include extending the security assessment to other types of attacks that exploit hierarchical structures within HFL, ensuring broader protection of federated learning systems.

A promising direction would be the integration of privacy-preserving techniques, such as differential privacy and secure multiparty computation, to balance model integrity with user privacy. Moreover, incorporating Explainable AI (XAI) methods could provide transparency into how adversarial attacks affect model updates, helping practitioners identify vulnerabilities and refine defense mechanisms dynamically.

Model Discrepancy Score and HFL Dynamics: Building on the work of studying HFL dynamics through the Model Discrepancy score, future research could refine and expand this metric to detect more subtle discrepancies in model updates, particularly in more complex, real-world federated learning deployments. Additional work could focus on making the discrepancy score more sensitive to evolving attack strategies that target federated learning models over time. Another important research direction could be the exploration of new aggregation methods that are more resistant to adversarial manipulation while maintaining the balance between system performance and security, with the discrepancy score playing a key role in detecting and mitigating adversarial updates during aggregation. Moreover, further research could explore how the interpretability of the discrepancy score impacts real-world decision-making systems, providing deeper insights that would enhance the robustness of Distributed DL models.

Further research could also examine how Model Discrepancy Scores can be used in real-time monitoring to create adaptive federated learning models that detect and respond to adversarial threats dynamically. Additionally, integrating reinforcement learning techniques may enable automated tuning of discrepancy thresholds, improving the model's ability to differentiate between natural variations and malicious updates.

Integration of Botnet Detection and HFL Security Assessment: An exciting area of future work involves the integration of the edge-based botnet detection system with the security assessment of HFL. By combining these two approaches, it may be possible to develop a unified framework that not only ensures the availability of Distributed DL systems by detecting botnet activities but also protects the integrity of federated learning models through real-time monitoring of model updates. This integrated system could leverage botnet detection at the edge to safeguard communication networks while simultaneously using the Model Discrepancy score to identify potential poisoning or adversarial attacks on the learning process. Such a combined approach would provide a multi-layered defense mechanism that strengthens the overall security of Distributed DL systems across the client-edge-cloud continuum.

One promising extension would be to design adaptive security policies that dynamically adjust defense mechanisms based on real-time threat assessments from both botnet detection and federated learning integrity monitoring. Moreover, exploring cross-layer security architectures that integrate IoT security, federated learning integrity, and AI-driven anomaly detection could provide a more comprehensive defense system tailored to diverse distributed DL applications.

6.4 Practical Implementations in IoT and Privacy-Sensitive Industries

The contributions of this research have significant practical applications in IoT-based systems and privacy-sensitive industries, addressing key security challenges in Distributed Deep Learning (DL) across the client-edge-cloud continuum. The proposed methodologies enhance both availability and integrity, making them highly relevant for real-world deployments.

1. Securing Smart Cities and Industrial IoT with EDIT The EDIT system offers a practical solution for securing IoT networks against early-stage botnet threats, which is crucial for smart city infrastructures, industrial IoT (IIoT), and autonomous systems. By leveraging edge-based detection, EDIT ensures real-time threat mitigation, preventing DDoS attacks that could disrupt critical services such as traffic management, smart grids, and connected healthcare systems. Its modular neural network (MNN) approach allows scalable deployment in resource-constrained IoT environments, ensuring efficient threat detection without overwhelming computational resources.

2. Strengthening Privacy-Centric AI in Healthcare and Finance The security assessment of Hierarchical Federated Learning (HFL) directly benefits privacy-sensitive industries, such as healthcare and finance, where data privacy is critical. By evaluating the vulnerabilities of HFL to adversarial attacks and proposing defense mechanisms like Adversarial Training (AT) and Neural Cleanse (NC), this research ensures secure federated model training without compromising data confidentiality. Hospitals and financial institutions can deploy federated learning models with stronger integrity, preventing data poisoning attacks that could lead to incorrect diagnoses or fraudulent transactions.

3. Enhancing AI Transparency and Trust in Critical Systems The integration of Explainable AI (XAI) techniques in both botnet detection (EDIT) and federated learning security (HFL) improves the trustworthiness of AI-driven decision-making in mission-critical applications. In cybersecurity, autonomous vehicles, and healthcare AI, understanding why a model detects a security threat or rejects a data update is essential for compliance and operational reliability. Future deployments of this research can integrate XAI-driven insights to help security analysts and AI practitioners better interpret attack patterns, model decisions, and anomaly detections in real-time environments.

4. Unified Security for Large-Scale Distributed AI Systems By integrating botnet detection (Chapter 3) with federated learning integrity mechanisms (Chapters 4 and 5), this research lays the foundation for a unified, multi-layered security framework for Distributed AI across IoT, edge, and cloud environments. This framework can be adopted in 5G-enabled

smart networks, defense and military AI systems, and secure cloud-edge collaboration platforms to ensure resilient AI deployments against evolving cyber threats.

References

[1] K. Agarwal, O. Khare, A. Sharma, A. Prakash, and A. K. Shukla, "Artificial Intelligence in a Distributed System of the Future," *Decentralized Systems and Distributed Computing*, pp. 317–335, 2024.

[2] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, and X. Shen, "Distributed artificial intelligence empowered by end-edge-cloud computing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 591–624, 2022.

[3] H. Gu, L. Zhao, Z. Han, G. Zheng, and S. Song, "AI-Enhanced Cloud-Edge-Terminal Collaborative Network: Survey, Applications, and Future Directions," *IEEE Communications Surveys & Tutorials*, 2023.

[4] A. Maia, A. Boutouchent, Y. Kardjadja, M. Gherari, E. G. Soyak, M. Saqib, K. Boussekar, I. Cilbir, S. Habibi, S. O. Ali, *et al.*, "A survey on integrated computing, caching, and communication in the cloud-to-edge continuum," *Computer Communications*, 2024.

[5] D. Rosendo, A. Costan, P. Valduriez, and G. Antoniu, "Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review," *Journal of Parallel and Distributed Computing*, vol. 166, pp. 71–94, 2022.

[6] Y. Wang, C. Yang, S. Lan, L. Zhu, and Y. Zhang, "End-edge-cloud collaborative computing for deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2024.

[7] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.

[8] Z. Xu, W. Liu, J. Huang, C. Yang, J. Lu, and H. Tan, "Artificial intelligence for securing IoT services in edge computing: a survey," *Security and communication networks*, vol. 2020, no. 1, p. 8872586, 2020.

[9] R. Mayer and H.-A. Jacobsen, "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–37, 2020.

[10] C. Prigent, A. Costan, G. Antoniu, and L. Cudennec, "Enabling federated learning across the computing continuum: Systems, challenges and future directions," *Future Generation Computer Systems*, 2024.

[11] O. Nassef, W. Sun, H. Purmehdi, M. Tatipamula, and T. Mahmoodi, "A survey: Distributed Machine Learning for 5G and beyond," *Computer Networks*, vol. 207, p. 108820, 2022.

[12] S. Ahmad, I. Shakeel, S. Mehfuz, and J. Ahmad, "Deep learning models for cloud, edge, fog, and IoT computing paradigms: Survey, recent advances, and future directions," *Computer Science Review*, vol. 49, p. 100568, 2023.

[13] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *Journal of Machine Learning Research*, vol. 20, no. 112, pp. 1–49, 2019.

[14] H. Choi, B. H. Lee, S. Y. Chun, and J. Lee, "Towards accelerating model parallelism in distributed deep learning systems," *Plos one*, vol. 18, no. 11, p. e0293338, 2023.

[15] D. Liu, X. Chen, Z. Zhou, and Q. Ling, "HierTrain: Fast hierarchical edge AI learning with hybrid parallelism in mobile-edge-cloud computing," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 634–645, 2020.

[16] M. Merluzzi, A. Martino, F. Costanzo, P. Di Lorenzo, and S. Barbarossa, "Dynamic ensemble inference at the edge," in *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2021.

[17] L. Kirsch, J. Kunze, and D. Barber, "Modular networks: Learning to decompose neural computation," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[18] H. Zhou, W. Zhang, C. Wang, X. Ma, and H. Yu, "Bbnet: a novel convolutional neural network structure in edge-cloud collaborative inference," *Sensors*, vol. 21, no. 13, p. 4494, 2021.

[19] Y. Gao, W. Wang, D. Wang, H. Wang, and Z. Zhang, "Cloud-edge inference under communication constraints: Data quantization and early exit," in *2022 International Symposium on Wireless Communication Systems (ISWCS)*, pp. 1–6, IEEE, 2022.

[20] A. Wainakh, A. S. Guinea, T. Grube, and M. Mühlhäuser, "Enhancing privacy via hierarchical federated learning," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 344–347, IEEE, 2020.

[21] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2020.

[22] X. Yu, L. Cherkasova, H. Vardhan, Q. Zhao, E. Ekaireb, X. Zhang, A. Mazumdar, and T. Rosing, "Async-HFL: Efficient and Robust Asynchronous Federated Learning in Hierarchical IoT Networks," in *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, IoTDI '23, (New York, NY, USA), p. 236–248, Association for Computing Machinery, 2023.

[23] H. Zhou, Y. Zheng, H. Huang, J. Shu, and X. Jia, "Toward Robust Hierarchical Federated Learning in Internet of Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[24] X. Zhou, X. Ye, K. I.-K. Wang, W. Liang, N. K. C. Nair, S. Shimizu, Z. Yan, and Q. Jin, "Hierarchical Federated Learning With Social Context Clustering-Based Participant Selection for Internet of Medical Things Applications," *IEEE Transactions on Computational Social Systems*, vol. 10, no. 4, pp. 1742–1751, 2023.

[25] Z. Wang, G. Yang, H. Dai, and C. Rong, "Privacy-Preserving Split Learning for Large-Scaled Vision Pre-Training," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1539–1553, 2023.

[26] H. Ko, B. Kim, Y. Kim, and S. Pack, "Two-Phase Split Computing Framework in Edge–Cloud Continuum," *IEEE Internet of Things Journal*, vol. 11, no. 12, pp. 21741–21749, 2024.

[27] R. Dong, Y. Mao, and J. Zhang, "Resource-Constrained Edge AI with Early Exit Prediction," *Journal of Communications and Information Networks*, vol. 7, no. 2, pp. 122–134, 2022.

[28] S. Yang, Z. Zhang, C. Zhao, X. Song, S. Guo, and H. Li, "CNNPC: End-Edge-Cloud Collaborative CNN Inference With Joint Model Partition and Compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4039–4056, 2022.

[29] H. Qi, F. Ren, L. Wang, P. Jiang, S. Wan, and X. Deng, "Multi-Compression Scale DNN Inference Acceleration based on Cloud-Edge-End Collaboration," *ACM Trans. Embed. Comput. Syst.*, vol. 23, jan 2024.

[30] B. Bala and S. Behal, "AI techniques for IoT-based DDoS attack detection: Taxonomies, comprehensive review and research challenges," *Computer science review*, vol. 52, p. 100631, 2024.

[31] C. Ma, J. Li, K. Wei, B. Liu, M. Ding, L. Yuan, Z. Han, and H. V. Poor, "Trusted AI in multiagent systems: An overview of privacy and security for distributed learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1097–1132, 2023.

[32] J. Zhang, P. Yu, L. Qi, S. Liu, H. Zhang, and J. Zhang, "FLDDoS: DDoS attack detection model based on federated learning," in *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 635–642, IEEE, 2021.

[33] R. Doriguzzi-Corin and D. Siracusa, "FLAD: adaptive federated learning for DDoS attack detection," *Computers & Security*, vol. 137, p. 103597, 2024.

[34] S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, and O. Jogunola, "Federated deep learning for zero-day botnet attack detection in IoT-edge devices," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3930–3944, 2021.

[35] K. P. Reddy, S. Kodati, M. Swetha, M. Parimala, and S. Velliangiri, "A hybrid neural network architecture for early detection of DDOS attacks using deep learning models," in *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*, pp. 323–327, IEEE, 2021.

[36] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "MIDeA: a multi-parallel intrusion detection architecture," in *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 297–308, 2011.

[37] G. Zhang, S. Lu, Y. Zhang, X. Chen, P.-Y. Chen, Q. Fan, L. Martie, L. Horesh, M. Hong, and S. Liu, "Distributed adversarial training to robustify deep neural networks at scale," in *Uncertainty in artificial intelligence*, pp. 2353–2363, PMLR, 2022.

[38] X. Li, Z. Song, and J. Yang, "Federated adversarial learning: A framework with convergence analysis," in *International Conference on Machine Learning*, pp. 19932–19959, PMLR, 2023.

[39] C. Chen, B. Kailkhura, R. Goldhahn, and Y. Zhou, "Certifiably-robust federated adversarial learning via randomized smoothing," in *2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, pp. 173–179, IEEE, 2021.

[40] D. Shah, P. Dube, S. Chakraborty, and A. Verma, "Adversarial training in communication constrained federated learning," *arXiv preprint arXiv:2103.01319*, 2021.

[41] Y. Zhao, Y. Cao, J. Zhang, H. Huang, and Y. Liu, "FlexibleFL: Mitigating poisoning attacks with contributions in cloud-edge federated learning systems," *Information Sciences*, vol. 664, p. 120350, 2024.

[42] Y. Zhou, R. Wang, X. Mo, Z. Li, and T. Tang, "Robust hierarchical federated learning with anomaly detection in cloud-edge-end cooperation networks," *Electronics*, vol. 12, no. 1, p. 112, 2022.

[43] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 707–723, IEEE, 2019.

[44] B. Liu, Kangand Dolan-Gavitt and S. Garg, "Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks," in *Research in Attacks, Intrusions, and Defenses* (M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, eds.), (Cham), pp. 273–294, Springer International Publishing, 2018.

[45] M. H. Meng, S. G. Teo, G. Bai, K. Wang, and J. S. Dong, "Enhancing federated learning robustness using data-agnostic model pruning," in *Advances in Knowledge Discovery and Data Mining* (H. Kashima, T. Ide, and W.-C. Peng, eds.), (Cham), pp. 441–453, Springer Nature Switzerland, 2023.

[46] C. Wu, X. Yang, S. Zhu, and P. Mitra, "Toward cleansing backdoored neural networks in federated learning," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pp. 820–830, IEEE, 2022.

[47] M. Usama, I. Ilahi, J. Qadir, R. N. Mitra, and M. K. Marina, "Examining Machine Learning for 5G and Beyond Through an Adversarial Lens," *IEEE Internet Computing*, vol. 25, no. 2, pp. 26–34, 2021.

[48] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning," in *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1354–1371, IEEE, 2022.

[49] O. Ibitoye, O. Shafiq, and A. Matrawy, "Analyzing Adversarial Attacks against Deep Learning for Intrusion Detection in IoT Networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2019.

[50] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *International Conference on Machine Learning*, pp. 634–643, PMLR, 2019.

[51] S. Li, E. C.-H. Ngai, and T. Voigt, "An Experimental Study of Byzantine-Robust Aggregation Schemes in Federated Learning," *IEEE Transactions on Big Data*, pp. 1–13, 2023.

[52] M. Bagaa, T. Taleb, J. B. Bernabe, and A. Skarmeta, "A machine learning security framework for iot systems," *IEEE Access*, vol. 8, pp. 114066–114077, 2020.

[53] M. Pawlicki, A. Pawlicka, R. Kozik, and M. Choraś, "The survey and meta-analysis of the attacks, transgressions, countermeasures and security aspects common to the Cloud, Edge and IoT," *Neurocomputing*, p. 126533, 2023.

[54] X. Guo, S. Han, X. S. Hu, X. Jiao, Y. Jin, F. Kong, and M. Lemmon, "Towards scalable, secure, and smart mission-critical IoT systems: review and vision," in *Proceedings of the 2021 International Conference on Embedded Software*, pp. 1–10, 2021.

[55] G. Abad, S. Picek, V. J. Ramírez-Durán, and A. Urbieta, "On the security & privacy in federated learning," *arXiv preprint arXiv:2112.05423*, 2021.

[56] M. Isakov, V. Gadepally, K. M. Gettings, and M. A. Kinsy, "Survey of attacks and defenses on edge-deployed neural networks," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, IEEE, 2019.

[57] R. Uddin, S. A. Kumar, and V. Chamola, "Denial of service attacks in edge computing layers: Taxonomy, vulnerabilities, threats and solutions," *Ad Hoc Networks*, vol. 152, p. 103322, 2024.

[58] A. Verma, R. Saha, N. Kumar, and G. Kumar, "A detailed survey of denial of service for IoT and multimedia systems: Past, present and futuristic development," *Multimedia Tools and Applications*, vol. 81, no. 14, pp. 19879–19944, 2022.

[59] N.-N. Dao, T. V. Phan, U. Sa'ad, J. Kim, T. Bauschert, D.-T. Do, and S. Cho, "Securing heterogeneous IoT with intelligent DDoS attack behavior learning," *IEEE Systems Journal*, vol. 16, no. 2, pp. 1974–1983, 2021.

[60] P. Kumari and A. K. Jain, "A comprehensive study of DDoS attacks over IoT network and their countermeasures," *Computers & Security*, vol. 127, p. 103096, 2023.

[61] A. Giannaros, A. Karras, L. Theodorakopoulos, C. Karras, P. Kranias, N. Schizas, G. Kalogeratos, and D. Tsolis, "Autonomous vehicles: Sophisticated attacks, safety issues, challenges, open topics, blockchain, and future directions," *Journal of Cybersecurity and Privacy*, vol. 3, no. 3, pp. 493–543, 2023.

[62] B. Sutheekshan, S. Basheer, G. Thangavel, and O. P. Sharma, "Evolution of Malware Targeting IoT Devices and Botnet formation," in *2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT)*, vol. 5, pp. 1415–1422, IEEE, 2024.

[63] M. Gelgi, Y. Guan, S. Arunachala, M. Samba Siva Rao, and N. Dragoni, "Systematic Literature Review of IoT Botnet DDOS Attacks and Evaluation of Detection Techniques," *Sensors*, vol. 24, no. 11, p. 3571, 2024.

[64] I. Ali, A. I. A. Ahmed, A. Almogren, M. A. Raza, S. A. Shah, A. Khan, and A. Gani, "Systematic literature review on IoT-based botnet attack," *IEEE access*, vol. 8, pp. 212220–212232, 2020.

[65] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, "Edge computing security: State of the art and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1608–1631, 2019.

[66] H. Karthikeyan and G. Usha, "Real-time DDoS flooding attack detection in intelligent transportation systems," *Computers and Electrical Engineering*, vol. 101, p. 107995, 2022.

[67] X. Xiao, Z. Tang, C. Li, B. Xiao, and K. Li, "SCA: Sybil-based collusion attacks of IIoT data poisoning in federated learning," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 3, pp. 2608–2618, 2022.

[68] V. G. Garagad, N. C. Iyer, and H. G. Wali, "Data integrity: a security threat for internet of things and cyber-physical systems," in *2020 International Conference on Computational Performance Evaluation (ComPE)*, pp. 244–249, IEEE, 2020.

[69] S. R. Dixit, P. N. Mahalle, and G. R. Shinde, "Rethinking Data Integrity in Federated Learning: Are we ready?," in *2022 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*, pp. 84–88, IEEE, 2022.

[70] C. Wang, J. Chen, Y. Yang, X. Ma, and J. Liu, "Poisoning attacks and countermeasures in intelligent networks: Status quo and prospects," *Digital Communications and Networks*, vol. 8, no. 2, pp. 225–234, 2022.

[71] K. N. Kumar, C. K. Mohan, and L. R. Cenkeramaddi, "The Impact of Adversarial Attacks on Federated Learning: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[72] J. Chen, X. Zhang, R. Zhang, C. Wang, and L. Liu, "De-pois: An attack-agnostic defense against data poisoning attacks," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3412–3425, 2021.

[73] Y. Wan, Y. Qu, W. Ni, Y. Xiang, L. Gao, and E. Hossain, "Data and model poisoning backdoor attacks on wireless federated learning, and the defense mechanisms: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2024.

[74] E. Rosenfeld, E. Winston, P. Ravikumar, and Z. Kolter, "Certified robustness to label-flipping attacks via randomized smoothing," in *International Conference on Machine Learning*, pp. 8230–8241, PMLR, 2020.

[75] W. Guo, B. Tondi, and M. Barni, "An overview of backdoor attacks against deep neural networks and possible defences," *IEEE Open Journal of Signal Processing*, vol. 3, pp. 261–287, 2022.

[76] S. Tufail, S. Batool, and A. I. Sarwat, "False data injection impact analysis in ai-based smart grid," in *SoutheastCon 2021*, pp. 01–07, IEEE, 2021.

[77] F. Wang, X. Wang, and X. J. Ban, "Data poisoning attacks in intelligent transportation systems: A survey," *Transportation Research Part C: Emerging Technologies*, vol. 165, p. 104750, 2024.

[78] Z. Wu, C. Xu, M. Wang, Y. Ma, Z. Wu, Z. Xiahou, and L. A. Grieco, "Patronus: Countering Model Poisoning Attacks in Edge Distributed DNN Training," in *2024 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, IEEE, 2024.

[79] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to {Byzantine-Robust} federated learning," in *29th USENIX security symposium (USENIX Security 20)*, pp. 1605–1622, 2020.

[80] Z. Wang, J. Ma, X. Wang, J. Hu, Z. Qin, and K. Ren, "Threats to training: A survey of poisoning attacks and defenses on machine learning systems," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–36, 2022.

[81] F. Aloraini, A. Javed, O. Rana, and P. Burnap, "Adversarial machine learning in IoT from an insider point of view," *Journal of Information Security and Applications*, vol. 70, p. 103341, 2022.

[82] A. Rahman, M. S. Hossain, N. A. Alrajeh, and F. Alsolami, "Adversarial examples—Security threats to COVID-19 deep learning systems in medical IoT devices," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9603–9610, 2020.

[83] J. Zhang and C. Li, "Adversarial examples: Opportunities and challenges," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 7, pp. 2578–2593, 2019.

[84] K. N. Kumar, C. Vishnu, R. Mitra, and C. K. Mohan, "Black-box adversarial attacks in autonomous vehicle technology," in *2020 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, pp. 1–7, IEEE, 2020.

[85] U. Ozbulak, M. Gasparyan, W. De Neve, and A. Van Messem, "Perturbation analysis of gradient-based adversarial attacks," *Pattern Recognition Letters*, vol. 135, pp. 313–320, 2020.

[86] Y. Bai, Y. Wang, Y. Zeng, Y. Jiang, and S.-T. Xia, "Query efficient black-box adversarial attack on deep neural networks," *Pattern Recognition*, vol. 133, p. 109037, 2023.

[87] R.-H. Hwang, J.-Y. Lin, S.-Y. Hsieh, H.-Y. Lin, and C.-L. Lin, "Adversarial patch attacks on deep-learning-based face recognition systems using generative adversarial networks," *Sensors*, vol. 23, no. 2, p. 853, 2023.

[88] B. Vignau, R. Khoury, S. Hallé, and A. Hamou-Lhadj, "The evolution of IoT Malwares, from 2008 to 2019: Survey, taxonomy, process simulator and perspectives," *Journal of Systems Architecture*, vol. 116, p. 102143, 2021.

[89] 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," *Computer Communications*, vol. 195, pp. 346–361, 2022.

[90] Z. Tian, L. Cui, J. Liang, and S. Yu, "A comprehensive survey on poisoning attacks and countermeasures in machine learning," *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–35, 2022.

[91] Z. You, D. Liu, B. Han, and C. Xu, "Beyond pretrained features: noisy image modeling provides adversarial defense," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[92] L. Zhu, C. Liu, Z. Zhang, Y. Cheng, B. Jie, and X. Ding, "An adversarial sample detection method based on heterogeneous denoising," *Machine Vision and Applications*, vol. 35, no. 4, p. 96, 2024.

[93] Y. Shi, M. Du, X. Wu, Z. Guan, J. Sun, and N. Liu, "Black-box backdoor defense via zero-shot image purification," *Advances in Neural Information Processing Systems*, vol. 36, pp. 57336–57366, 2023.

[94] Y. Wang, X. Ma, J. Bailey, J. Yi, B. Zhou, and Q. Gu, "On the Convergence and Robustness of Adversarial Training," in *International Conference on Machine Learning*, pp. 6586–6595, PMLR, 2019.

[95] A. Shafahi, M. Najibi, M. A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein, "Adversarial training for free!," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[96] Q. Han, S. Lu, W. Wang, H. Qu, J. Li, and Y. Gao, "Privacy preserving and secure robust federated learning: A survey," *Concurrency and Computation: Practice and Experience*, p. e8084, 2024.

[97] H. Zhou, Y. Zheng, and X. Jia, "Towards robust and privacy-preserving federated learning in edge computing," *Computer Networks*, vol. 243, p. 110321, 2024.

[98] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artificial Intelligence Review*, vol. 53, pp. 5113–5155, 2020.

[99] B. Li, P. Qi, B. Liu, S. Di, J. Liu, J. Pei, J. Yi, and B. Zhou, "Trustworthy AI: From principles to practices," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–46, 2023.

[100] M. Wazzan, D. Algazzawi, O. Bamasaq, A. Albeshri, and L. Cheng, "Internet of Things botnet detection approaches: Analysis and recommendations for future research," *Applied Sciences*, vol. 11, no. 12, p. 5713, 2021.

[101] G. Kambourakis, M. Anagnostopoulos, W. Meng, and P. Zhou, *Botnets: Architectures, countermeasures, and challenges*. CRC Press, 2019.

[102] R. Vishwakarma and A. K. Jain, "A survey of DDoS attacking techniques and defence mechanisms in the IoT network," *Telecommunication systems*, vol. 73, no. 1, pp. 3–25, 2020.

[103] N. Vlajic and D. Zhou, "IoT as a Land of Opportunity for DDoS Hackers," *Computer*, vol. 51, no. 7, pp. 26–34, 2018.

[104] S. Dange and M. Chatterjee, "IoT botnet: the largest threat to the IoT network," in *Data Communication and Networks*, pp. 137–157, Springer, 2020.

[105] Y. Jia, F. Zhong, A. Alrawais, B. Gong, and X. Cheng, "FlowGuard: An Intelligent Edge Defense Mechanism Against IoT DDoS Attacks," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9552–9562, 2020.

[106] M. M. Alani, "BotStop : Packet-based efficient and explainable IoT botnet detection using machine learning," *Computer Communications*, vol. 193, pp. 53–62, 2022.

[107] M. Hegde, G. Kepnang, M. Al Mazroei, J. S. Chavis, and L. Watkins, "Identification of botnet activity in IoT network traffic using machine learning," in *2020 International conference on intelligent data science technologies and applications (IDSTA)*, pp. 21–27, IEEE, 2020.

[108] H. Alzahrani, M. Abulkhair, and E. Alkayal, "A multi-class neural network model for rapid detection of IoT botnet attacks," *Int. J. Adv. Comp. Sci. Appl*, vol. 11, no. 7, pp. 688–696, 2020.

[109] N. Abdalgawad, A. Sajun, Y. Kaddoura, I. A. Zualkernan, and F. Aloul, "Generative deep learning to detect cyberattacks for the iot-23 dataset," *IEEE Access*, vol. 10, pp. 6430–6441, 2021.

[110] M. Wazzan, D. Algazzawi, A. Albeshri, S. Hasan, O. Rabie, and M. Z. Asghar, "Cross Deep Learning Method for Effectively Detecting the Propagation of IoT Botnet," *Sensors*, vol. 22, no. 10, p. 3895, 2022.

[111] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, and S. Nõmm, "MedBIoT: Generation of an IoT Botnet Dataset in a Medium-sized IoT Network.," in *ICISSP*, pp. 207–218, 2020.

[112] L. Giaretta, A. Lekssays, B. Carminati, E. Ferrari, and Š. Girdzijauskas, "LiMNet: Early-Stage Detection of IoT Botnets with Lightweight Memory Networks," in *European Symposium on Research in Computer Security*, pp. 605–625, Springer, 2021.

[113] R. Gandhi and Y. Li, "Comparing Machine Learning and Deep Learning for IoT Botnet Detection," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 234–239, IEEE, 2021.

[114] K. Malik, F. Rehman, T. Maqsood, S. Mustafa, O. Khalid, and A. Akhunzada, "Lightweight Internet of Things Botnet Detection Using One-Class Classification," *Sensors*, vol. 22, no. 10, p. 3646, 2022.

[115] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto, and K. Sakurai, "A sequential scheme for detecting cyber attacks in IoT environment," in *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pp. 238–244, IEEE, 2019.

[116] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto, and K. Sakurai, "Machine learning-based IoT-botnet attack detection with sequential architecture," *Sensors*, vol. 20, no. 16, p. 4372, 2020.

[117] F. Hussain, S. G. Abbas, I. M. Pires, S. Tanveer, U. U. Fayyaz, N. M. Garcia, G. A. Shah, and F. Shahzad, "A Two-Fold Machine Learning Approach to Prevent and Detect IoT Botnet Attacks," *IEEE Access*, vol. 9, pp. 163412–163430, 2021.

[118] G. L. Nguyen, B. Dumba, Q.-D. Ngo, H.-V. Le, and T. N. Nguyen, "A collaborative approach to early detection of IoT Botnet," *Computers & Electrical Engineering*, vol. 97, p. 107525, 2022.

[119] A. Kumar, M. Shridhar, S. Swaminathan, and T. J. Lim, "Machine learning-based early detection of IoT botnets using network-edge traffic," *Computers & Security*, vol. 117, p. 102693, 2022.

[120] K. Chen, "Deep and modular neural networks," in *Springer Handbook of Computational Intelligence*, pp. 473–494, Springer, 2015.

[121] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," in *26th USENIX security symposium (USENIX Security 17)*, pp. 1093–1110, 2017.

[122] J. M. Ceron, K. Steding-Jessen, C. Hoepers, L. Z. Granville, and C. B. Margi, "Improving iot botnet investigation using an adaptive network layer," *Sensors*, vol. 19, no. 3, p. 727, 2019.

[123] J. Gamblin, "Mirai Source-Code. 2017," 2019.

[124] X. Zhang, O. Upton, N. L. Beebe, and K.-K. R. Choo, "IoT botnet forensics: A comprehensive digital forensic case study on Mirai botnet servers," *Forensic Science International: Digital Investigation*, vol. 32, p. 300926, 2020.

[125] A. Parmisano, S. Garcia, and M. Erquiaga, "Iot-23 dataset: A labeled dataset of malware and benign IoT traffic," *Avast-AIC laboratory, Stratosphere IPS, Czech Technical University (CTU)*, 2019.

[126] B.-L. Lu and M. Ito, "Task decomposition and module combination based on class relations: a modular neural network for pattern classification," *IEEE Transactions on Neural networks*, vol. 10, no. 5, pp. 1244–1256, 1999.

[127] B. R. Silva, R. J. Silveira, M. G. da Silva Neto, P. C. Cortez, and D. G. Gomes, "A comparative analysis of undersampling techniques for network intrusion detection systems design," *Journal of Communication and Information Systems*, vol. 36, no. 1, pp. 31–43, 2021.

[128] P. Bedi, N. Gupta, and V. Jindal, "Siam-IDS: Handling class imbalance problem in intrusion detection systems using siamese neural network," *Procedia Computer Science*, vol. 171, pp. 780–789, 2020.

[129] J. YAN, T. CHEN, B. XIE, Y. SUN, S. ZHOU, and Z. NIU, "Hierarchical Federated Learning: Architecture, Challenges, and Its Implementation in Vehicular Networks," *ZTE Communications*, vol. 21, no. 1, pp. 38–45, 2023.

[130] O. Rana, T. Spyridopoulos, N. Hudson, M. Baughman, K. Chard, I. Foster, and A. Khan, "Hierarchical and decentralised federated learning," in *2022 Cloud Continuum*, pp. 1–9, IEEE, 2022.

[131] D.-J. Han, M. Choi, J. Park, and J. Moon, "Fedmes: Speeding up federated learning with multiple edge servers," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3870–3885, 2021.

[132] Z. Qu, X. Li, J. Xu, B. Tang, Z. Lu, and Y. Liu, "On the convergence of multi-server federated learning with overlapping area," *IEEE Transactions on Mobile Computing*, 2022.

[133] M. P.-L. Ooi, S. Sohail, V. G. Huang, N. Hudson, M. Baughman, O. Rana, A. Hinze, K. Chard, R. Chard, I. Foster, *et al.*, "Measurement and applications: Exploring the challenges and opportunities of hierarchical federated learning in sensor applications," *IEEE Instrumentation & Measurement Magazine*, vol. 26, no. 9, pp. 21–31, 2023.

[134] A. E. Abyane, D. Zhu, R. Souza, L. Ma, and H. Hemmati, "Towards understanding quality challenges of the federated learning for neural networks: a first look from the lens of robustness," *Empirical Software Engineering*, vol. 28, no. 2, p. 44, 2023.

[135] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Hierarchical federated learning with quantization: Convergence analysis and system design," *IEEE Transactions on Wireless Communications*, vol. 22, no. 1, pp. 2–18, 2022.

[136] N. Al-Maslamani, M. Abdallah, and B. S. Ciftler, "Reputation-Aware Multi-Agent DRL for Secure Hierarchical Federated Learning in IoT," *IEEE Open Journal of the Communications Society*, 2023.

[137] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, *et al.*, "Adversarial Robustness Toolbox v1. 0.0," *arXiv preprint arXiv:1807.01069*, 2018.

[138] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[139] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[140] Z. Zheng, Y. Zhou, Y. Sun, Z. Wang, B. Liu, and K. Li, "Applications of federated learning in smart cities: recent advances, taxonomy, and open challenges," *Connection Science*, vol. 34, no. 1, pp. 1–28, 2022.

[141] H. Zhu, J. Xu, S. Liu, and Y. Jin, "Federated learning on non-IID data: A survey," *Neurocomputing*, vol. 465, pp. 371–390, 2021.

[142] S. Lee, A. K. Sahu, C. He, and S. Avestimehr, "Partial model averaging in federated learning: Performance guarantees and benefits," *Neurocomputing*, vol. 556, p. 126647, 2023.

[143] G. Xia, J. Chen, C. Yu, and J. Ma, "Poisoning attacks in federated learning: A survey," *IEEE Access*, vol. 11, pp. 10708–10722, 2023.

[144] L. Lyu, H. Yu, X. Ma, C. Chen, L. Sun, J. Zhao, Q. Yang, and S. Y. Philip, "Privacy and robustness in federated learning: Attacks and defenses," *IEEE transactions on neural networks and learning systems*, 2022.

[145] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," *arXiv preprint arXiv:2012.13995*, 2020.

[146] E. Isik-Polat, G. Polat, and A. Kocyigit, "ARFED: Attack-Resistant Federated averaging based on outlier elimination," *Future Generation Computer Systems*, vol. 141, pp. 626–650, 2023.

[147] S. Li, E. Ngai, and T. Voigt, "Byzantine-Robust Aggregation in Federated Learning Empowered Industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1165–1175, 2023.

[148] S. Luo, X. Chen, Q. Wu, Z. Zhou, and S. Yu, "HFEL: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning," *IEEE Transactions on Wireless Communications*, vol. 19, no. 10, pp. 6535–6548, 2020.

[149] X. Xie, C. Hu, H. Ren, and J. Deng, "A survey on vulnerability of federated learning: A learning algorithm perspective," *Neurocomputing*, p. 127225, 2024.

[150] Q. Xia, Z. Tao, Z. Hao, and Q. Li, "FABA: an algorithm for fast aggregation against byzantine attacks in distributed neural networks," in *IJCAI*, 2019.

[151] A. Qayyum, M. U. Janjua, and J. Qadir, "Making federated learning robust to adversarial attacks by learning data and model association," *Computers & Security*, vol. 121, p. 102827, 2022.

[152] C. Yin and Q. Zeng, "Defending against data poisoning attack in federated learning with non-iid data," *IEEE Transactions on Computational Social Systems*, 2023.

[153] A. Yazdinejad, A. Dehghantanha, H. Karimipour, G. Srivastava, and R. M. Parizi, "A robust privacy-preserving federated learning model against model poisoning attacks," *IEEE Transactions on Information Forensics and Security*, 2024.

[154] A. Gupta, T. Luo, M. V. Ngo, and S. K. Das, "Long-short history of gradients is all you need: Detecting malicious and unreliable clients in federated learning," in *European Symposium on Research in Computer Security*, pp. 445–465, Springer, 2022.

[155] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2351–2363, 2020.