Privacy and Security Preserving Distributed Machine Learning and Data Storage in Large-Scale Internet of Things

TIANTONG WU

B.Eng (Hons I), MDS

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

School of Electrical and Computer Engineering
Faculty of Engineering
The University of Sydney
Australia

8 April 2025

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is the result of my original work during the candidature of my higher degree by research. This thesis does not include a substantial part of work that has been submitted for any other degree or diploma in any institution.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged. Where I have consulted the published work of others, this is always clearly attributed.

Tiantong Wu
29 August 2024

Authorship Attribution

The publications used in this thesis and details of my contributions are listed below:

- Chapter 3 of this thesis is published as [J1], which is the extended version of [C1].
  I contributed to the literature survey and design of the dual-blockchain framework and its corresponding detailed algorithm, and conducted the theoretical analysis for the proposed framework. I developed the experimental testbed, conducted the experiments, recorded and analysed the results.
- Chapter 4 of this thesis is published as [C3], and the extended version that is currently in preparation as [J2].
  I contributed to literature survey and design of the system model and its corresponding algorithms. I conducted theoretical analysis and mathematical proofs. I developed the experimental testbed, conducted the experiments, recorded and analysed the experimental results.
- Chapter 5 of this thesis is published as [C2].
  I contributed to the literature survey and design of the system architecture and the algorithms. I conducted theoretical analysis and simulation-based experiments. I recorded and analysed the results from the experiments.
- Chapter 6 of this thesis is currently in preparation as [J3].
  I contributed to the literature survey and design of the system model. I developed the architecture and deployed the experimental testbed. I recorded and analysed the results from the experiments.
- Chapter 7 of this thesis is currently in preparation as [C4].
  I contributed to the literature survey and design of the system model. I performed theoretical analysis. I deployed experimental testbed, recorded and analysed the experimental results.

In addition to the statements above, in cases where I am not the corresponding author of a published item, permission to include the published material has been granted by the corresponding author.

Tiantong Wu

29 August 2024

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Teng Joon Lim

30 August 2024

Abstract

With the rapid growth of the Internet of Things (IoT) devices, managing and processing the enormous volume of IoT data has become a significant challenge. Typically, training a Machine Learning (ML) model to a high utility requires a large number of data samples. Therefore, the limited number of data samples collected by a single client device is insufficient to utilise ML-based methods to process the local data effectively. To address the data island effect caused by the isolation of the client devices, distributed storage frameworks are proposed so that the IoT data can be stored at multiple clients in a decentralised manner. While distributed data storage improves efficiency and enables many use cases, it incurs trust and security challenges. To improve efficiency while maintaining trust and security in distributed data storage, this thesis first presents a blockchain-based distributed storage framework named ***MapChain-D***. *MapChain-D* includes two mapped blockchains located at different groups of devices for data storage and indexing.

After collecting and storing the IoT data, ML-based methods can be used to process the data and explore more insightful patterns of the data collected. Distributed Machine Learning (DML) frameworks such as Federated Learning (FL) and Split Learning (SL) have been proposed to train a fully- or partially-shared model collaboratively by multiple participants (e.g., clients and edge servers) to improve privacy and efficiency. However, trust issues have become a major concern in DML because different clients or edge servers might attempt to change the model parameters or aggregation weights maliciously. To mitigate such model integrity attacks, this thesis then presents a blockchain- and homomorphic hash-based model verification technique named ***VHFL***. *VHFL* efficiently verifies the integrity of model aggregation at different hierarchical levels in a Hierarchical FL (HFL) network.

Apart from integrity issues, recent research has highlighted potential privacy concerns from the trained model parameters shared in DML networks. Furthermore, existing DML

frameworks do not allow clients to set their personalised privacy levels based on their preferences for model sharing. To overcome this challenge, we proposed two personalised privacy preservation techniques for DML, named ***FlexSplit*** and ***FlexMask***, aiming to provide clients with more flexible options to participate in DML whilst setting their preferred level of model sharing and preserving original model utility. Specifically, *FlexSplit* allows each client to select a layer to be shared with the edge server. In contrast, *FlexMask* enables each client to determine a personalised mask that selects different neurons in a layer to hide a specific attribute.

In a DML network, different clients might also be interested in various training tasks. The final part of this thesis presents a personalised neuron selection technique, named ***FlexNS***, for each DML client to select neurons in a pre-trained model and train a personalised lightweight model according to the client's local training task and private data.

Acknowledgements

My sincere gratitude to my lead supervisor, Prof Teng Joon Lim, for his priceless support and guidance. His constructive comments and suggestions have significantly improved my research work.

I am equally thankful to my associate supervisors, Dr Kanchana Thilakarathna and Dr Phee Lep Yeoh, for their persistent advice and encouragement to enhance my scientific knowledge and research skills throughout my PhD candidature.

I thank my Data61 co-supervisor, Dr Dilum Bandara, for his valuable mentoring and continuous support in improving my papers. I thank Dr Guillaume Jourjon for his constructive suggestions during the beginning of my PhD journey.

I extend my thanks to all my colleagues at the University of Sydney and Data61 CSIRO.

I acknowledge the financial support through the Data61 Postgraduate Scholarship and the University of Sydney Faculty of Engineering International Research Scholarship provided by Data61 CSIRO and the University of Sydney. I acknowledge the funding provided by Prof Teng Joon Lim, Dr Phee Lep Yeoh and Dr Kanchana Thilakarathna for the computational hardware costs and travel expenses for conferences and research visits.

Last but not least, to my dearest mother, Muxin Yang and father, Haohua Wu, I could not have finished my PhD without your unconditional support and encouragement. Words cannot express my love and appreciation towards you.

List of Publications

Publications Used in Thesis

Journal Articles

[J1] **Tiantong Wu**, Guillaume Jourjon, Kanchana Thilakarathna and Phee Lep Yeoh, "MapChain-D: A Distributed Blockchain for IIoT Data Storage and Communications," in IEEE Transactions on Industrial Informatics, vol. 19, no. 9, pp. 9766-9776, Sept. 2023, doi: 10.1109/TII.2023.3234631.

[J2] **Tiantong Wu**, H.M.N. Dilum Bandara, Kanchana Thilakarathna and Phee Lep Yeoh, "Verifiable HFL: Mitigating Model Aggregation Attacks in Hierarchical Federated Learning," submitted to Transactions on Emerging Telecommunications Technologies.

[J3] **Tiantong Wu**, H.M.N. Dilum Bandara, Kanchana Thilakarathna, Phee Lep Yeoh and Teng Joon Lim, "FlexMask: Personalised Masking for Property Inference Attack Protection in Split Learning," submitted to IEEE Transactions on Knowledge and Data Engineering.

Conference Proceedings

[C1] **Tiantong Wu**, Phee Lep Yeoh, Guillaume Jourjon and Kanchana Thilakarathna, "MapChain: A DHT-Based Dual-Blockchain Data Structure for Large-Scale IoT Systems," 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), New Orleans, LA, USA, 2021, pp. 177-182, doi: 10.1109/WF-IoT51360.2021.9595910.

[C2] **Tiantong Wu**, H.M.N. Dilum Bandara, Phee Lep Yeoh and Kanchana Thilakarathna, "FlexSplit: A Configurable, Privacy-Preserving Federated-Split Learning Framework," 2023 IEEE International Conference on Communications Workshops (ICC

Workshops), Rome, Italy, 2023, pp. 116-121, doi: 10.1109/ICCWorkshops57953.2023.10283667.

[C3] **Tiantong Wu**, H.M.N. Dilum Bandara, Phee Lep Yeoh and Kanchana Thilakarathna, "VHFL: A Cloud-Edge Model Verification Technique for Hierarchical Federated Learning," 2024 IEEE International Conference on Communications Workshops (ICC Workshops), Denver, CO, USA, 2024, pp. 1304-1309, doi: 10.1109/ICCWorkshops59551.2024.10615330.

[C4] **Tiantong Wu**, H.M.N. Dilum Bandara, Kanchana Thilakarathna, Phee Lep Yeoh and Teng Joon Lim, "FlexNS: Communication- and Computation-Efficient Edge Learning for Multi-Task Transfer Learning in AIoT", submitted to IEEE International Conference on Pervasive Computing and Communications (PerCom).

Other Publication during Candidature

Conference Proceeding

[C5] Xin Hao, Phee Lep Yeoh, **Tiantong Wu**, Yao Yu, Yonghui Li and Branka Vucetic, "Scalable Double Blockchain Architecture for IoT Information and Reputation Management," 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), New Orleans, LA, USA, 2021, pp. 171-176, doi: 10.1109/WF-IoT51360.2021.9595791.

Contents

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
BFT	Byzantine Fault Tolerant
CHS	Calinski-Harabasz Score
CNN	Convolutional Neural Network
DBS	Davies-Bouldin Score
DHT	Distributed Hash Table
DLT	Distributed Ledger Technology
DML	Distributed Machine Learning
DP	Differential Privacy
DRA	Data Reconstruction Attack
FL	Federated Learning
HFL	Hierarchical Federated Learning
IID	Independent and Identically Distributed
IoT	Internet of Things
JSON	JavaScript Object Notation
GAN	Generative Adversarial Network
MI	Mutual Information
MIA	Membership Inference Attack
ML	Machine Learning
P2P	Peer-to-Peer
PIA	Property Inference Attack
PoA	Proof-of-Authority
PoW	Proof-of-Work

SL Split Learning

SLP Single-Layer Perception

TL Transfer Learning

TCP Transmission Control Protocol

TEE Trusted Execution Environment

List of Symbols and Notations

C	Centroid
D	Number of data samples
G	Number of groups
H	Hash value
L	Number of parameters in a layer
N	Number of clients/nodes
R	Number of neurons in a layer
S	Storage size
W	Between-cluster dispersion
d	Data sample index (the d^{th} sample)
e	Client weight
g	Group index (the g^{th} group)
k	Number of selected neurons
l	Layer index (the l^{th} layer)
m	Client index (the m^{th} client)
n	Size of client's local dataset
p	Queue length
r	Neuron index (the r^{th} neuron)
t	Time
y	Classification label
μ	Mean
σ	Standard deviation
fs	CNN layer filter size
nc	Number of convolutional layers in a CNN

nf	Number of fully-connected layers in a CNN
ni	Number of input features
no	Number of classes
nu	Number of model updates
os	CNN layer output dimension
c	Activation outputs from a layer
e	Activation outputs from a neuron
f	Input features
l	Neuron mask labels
w	Model parameters
y	Classification labels for a set of samples
$\mathcal{B}$	Block in a blockchain
$\mathcal{C}$	Blockchain
$\mathcal{D}$	End device/Client
$\mathcal{E}$	Edge server
$\mathcal{M}$	Data frame
$\mathcal{S}$	Device-specific information
ASR	Attack Success Rate
IVS	Integrity Verification Sensitivity
VA	Validation Accuracy

List of Figures

List of Tables

CHAPTER 1

Introduction

The rapid growth of the Internet of Things (IoT) and mobile devices contributes to the majority of the projected 41 billion devices connected to the Internet by 2025 [1] and 50 billion by 2030. The majority of the IoT data is continuously collected by the mass deployment of sensors in home automation, smart metering and wearable applications [2]. Due to cost limitations for a single device in large-scale deployments of IoT networks, a large portion of the IoT edge devices are constrained in computational and storage resources, and they typically have strict power limits due to the long-term operational and scalability requirements. As the volume of the IoT data increases explosively due to the increasing number of IoT sensors, scalability and efficiency have become essential requirements for IoT data storage and management [3]. Therefore, the IoT data should be maintained by devices with larger storage space, and the data should be accessible by devices capable of processing it.

As illustrated in Fig. 1.1, the conventional approach of deploying an IoT network is to adopt a "star" topology where all the resource-constrained IoT edge devices (i.e., clients) that collect the data are connected to a trusted centralised service provider with significantly more computational and storage resources (i.e., cloud server) to process the data received. All clients transfer the collected data to a cloud server for further processing. However, it is infeasible for a large-scale IoT network to transmit all the raw data collected by a large number of client devices to a single cloud server as it creates a communication bottleneck due to an extensive volume of many-to-one network traffic to the cloud server. Centralised approaches also introduce a single point of failure due to the entire network relying on a single cloud server. Additionally, centralised approaches incur privacy issues during data transmission and retrieval processes because the raw data collected by the client devices is shared with

other entities in the system. To mitigate such issues, distributed storage and computing, which performs storage and computation at multiple IoT devices in a decentralised manner, has become an active area of research.

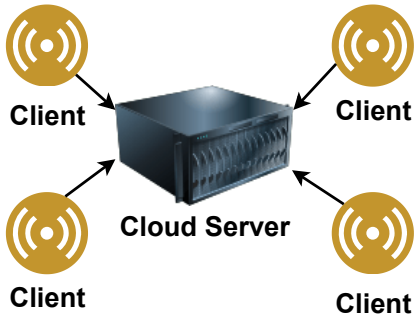


FIGURE 1.1. Star Topology

Peer-to-Peer (P2P) storage is a widely used approach in IoT distributed data storage to mitigate the communications and storage bottlenecks in conventional cloud-based data storage by introducing multiple edge storage devices to offload the resource overheads at the remote cloud server. In a P2P storage system, the IoT data is stored among multiple edge devices of the network rather than a single remote centralised server. The devices that are responsible for data storage are interconnected to propagate the data in the network. Adopting P2P storage in IoT can reduce the single point of failure at the cloud server, as well as privacy leakage to the centralised cloud service provider [4]. In addition, the efficiency of data insertion and retrieval is improved due to the parallel data processing and transfer among multiple IoT edge devices [4], making this solution more scalable in IoT scenarios.

After data collection and storage, the next step is to process the raw data to extract more insightful information from it. Machine Learning (ML) is a promising technique for performing Artificial Intelligence (AI) tasks where a statistical model is trained to perform various tasks (i.e., prediction or inference) without needing to create instructions and rules. An Artificial Neural Network (ANN) is a type of advanced ML model inspired by the biological neural network [5]. A typical ANN model consists of multiple layers of connected neurons. It requires a large amount of data samples to train an ANN model to a high utility. Therefore, it is impractical for each edge device to train an individual model separately using only their local data.

The conventional centralised approach of training an ANN model in IoT systems requires all clients to upload their raw data to the cloud server, and the shared global ANN model is trained by the cloud server that has stronger computing power. In large-scale IoT networks, cloud-based ML model training is infeasible due to the large volume of data and large amount of IoT devices in the system. It shares the same drawbacks of cloud-based data storage, including efficiency issues (i.e., communication bottleneck, single point of failure, and privacy leakage) and privacy concerns caused by sharing the data with an external party. Additionally, cloud-based training causes a computational bottleneck because a single global ANN model needs to be trained in serial using data samples from all clients. Distributed Machine Learning (DML) is a promising solution to address those drawbacks by training a shared ANN model collaboratively at all multiple participants (e.g., clients, edge servers) that hold the raw data. In DML, each participant performs local training of an ANN model using the raw data it holds and shares the locally trained model parameters with other participants in the network. Instead of performing all the training processes, the cloud server in a DML network is only responsible for coordinating the clients or performing partial training processes offloaded from the clients. Similar to the advantages of distributed data storage, DML reduces the dependency on the cloud server, which weakens the dominance of a single entity, reduces the single point of failure and improves efficiency.

1.1 Motivation

Due to the interconnected nature of the distributed IoT networks, the security weak point depends on the most vulnerable IoT device [2]. Therefore, data integrity and confidentiality should be enforced across the entire IoT network. Typically, edge devices in an IoT network belong to multiple users or organisations with different privacy and security needs. They might be controlled by malicious users or organisations with competitive businesses [2]. Therefore, while distributed data storage improves the scalability, efficiency and reliability of data storage in IoT systems, moving data storage to the edge might cause data privacy, security and trust issues [4], which include:

- Privacy leakage: the data might be stored on insecure edge devices, which causes privacy leakage to an external party. The data might be transferred within insecure networks and eavesdropped by an external party if unencrypted.
- Data availability: data transfer could be compromised by malicious participants or a third party, making it inaccessible or unusable. Malicious participants might selectively respond to data retrieval requests from a subset of the requesters while ignoring other requests or only responding to a subset of requests.
- Data integrity: the data stored could be tampered with by malicious participants upon their interests, which means that the data is still accessible, but the contents are different from the original. Malicious participants might respond differently to the same data retrieval requests from different requesters or at different times.

Similar to the distributed data storage, recent research also shows that moving the ML training process to the edge might lead to the following trust and security concerns [6], [7]:

- Privacy leakage: an adversary might perform privacy attacks (e.g., Data Reconstruction Attack (DRA), Property Inference Attack (PIA) and Membership Inference Attack (MIA)) to infer clients' raw data or properties using the model parameters transferred between clients and the server.
- Malicious models: a malicious client or server can train the model using poisoned datasets to lead the global model towards a direction of their interest or reduce the global model utility.

Apart from trust and security issues, efficiency issues have also been considered [7] in DML systems due to the recent rapid increase in model size. One of the hurdles of deploying DML in resource-constrained edge IoT devices is training a complex model on edge devices with lower computing capability. The diversity of the training tasks of different clients also increases the model complexity and training difficulties. Therefore, training a DML model efficiently with optimisations on both resource consumption and model utility has become an important problem when deploying a DML model on large-scale IoT systems.

1.1.1 Challenges in IoT Privacy and Security

This thesis aims to provide solutions for privacy and security in large-scale IoT distributed machine learning and data storage. In large-scale IoT systems, the unavoidable heterogeneous nature of the IoT systems increases the difficulty of privacy and security protection [8]. Particularly, the trust and security levels, as well as the computation and communications capabilities between different participants, are different for various IoT devices. Therefore, it is crucial to ensure that participants with lower computation and communications capabilities are able to access the data and verify its integrity without causing additional bottlenecks in the system. While existing approaches consider privacy, trust and security preservation in distributed data storage and DML, there is a lack of solutions that consider privacy, trust and security in IoT heterogeneous systems. In particular, the heterogeneities include:

- Privacy and trust levels: different end users or organisations have diverse privacy requirements and sensitive information they wish to protect. Different participants have various trust levels and might be deployed within physical and networking environments with various trust levels.
- Computation and communication resources: different participants in the network have various limitations in computing and communication resources. They might have different latencies when responding to queries or performing computation tasks.
- Performance requirements: participants might vary in application performance requirements, such as data availability for data storage applications and model accuracy for machine learning-based applications. Some applications are critical, require all data to be available at any time, and have higher accuracy, whereas others have more tolerance in performance.
- Objectives: different participants might have different objectives. For example, in a distributed storage network, some participants might be interested in retrieving a particular type of data; in a DML network, different participants are interested in different training tasks.

Due to the above heterogeneity of client devices' capabilities and requirements, it becomes important to consider the variations of their capabilities and requirements [8]. In practical large-scale IoT systems, it is infeasible to update privacy and security policies frequently, due to the communication constraints. Therefore, autonomous and self-adaptive solutions are desirable.

1.1.2 Measurements and Metrics

To assess the efficiency of the frameworks proposed in the thesis under the IoT environment, the following performance measurements and metrics are used:

- CPU and memory usage: To measure the computing resource usage, CPU and memory usage are recorded. CPU usage can be recorded as the percentage of time when the central processing unit (CPU) is active or carrying out computation operations. Memory usage is the amount of random access memory (RAM) used to store active data or instructions. The CPU and memory usage is critical for IoT applications because most of the IoT devices are limited in computational resources.
- Network usage and latency: To measure the communications performance, the network usage and latency are recorded. Network usage is the volume of data required to perform an operation, whereas latency is the time taken to complete an operation. Those measurements are important for practical IoT deployment because of the bandwidth limitations of large-scale IoT network environment and the requirement of latency and freshness of IoT data.
- Computing time overhead: The computing time overhead is measured as the additional time required to perform a task. The overall computing time overhead is calculated as the difference between the overall time consumption with and without the components added to the original algorithm. The computing time overhead is an important measurement for evaluating the practicability and feasibility of the proposed algorithms in a resource-constrained IoT environment.

To assess the scalability of the proposed solution, the measurements and metrics are recorded for different numbers of IoT clients and the trend is analysed. Those measurements are used in the practical hardware-based emulations in this thesis.

In addition to the performance measurements and metrics, the effectiveness of the proposed algorithms can be measured using the following criteria:

- Integrity Verification Sensitivity (IVS): IVS measures the effectiveness of the model parameter verification results. It is defined as the ratio of the correct results and the total number of results. The detailed description of the IVS calculation is presented in Sec. 4.4.4.
- Model Accuracy (MA): MA is the trained model's accuracy on the validation dataset. It is defined as the ratio of the correct results and the total number of data samples in the validation dataset.
- Privacy Measurements: The volume of private information embedded in part of a model. This can be measured as the SHapley Additive exPlanations (SHAP) value, Rand index, mutual information (MI) and attack success rate (ASR).

1.2 Contributions

In this thesis, various approaches for personalised distributed data storage and DML are proposed mainly focusing on privacy and security aspects. The proposed approaches integrate commonly used theorems, algorithms and paradigms to address the unresolved challenges. This ensures that the proposed solutions are generalisable under the majority of the practical use cases. The proposed approaches are analysed theoretically and experimentally using simulation or emulation methods. Raspberry Pi (RPi)- and Arduino-based testbeds are deployed to simulate real-life IoT sensors and client devices.

The contributions of this thesis can be divided into two main perspectives, including **distributed data storage** and **DML**. As depicted in Fig.1.2, five techniques (in green colour boxes) are proposed in Chapters 3-7 to provide personalised integrity, security and privacy protection

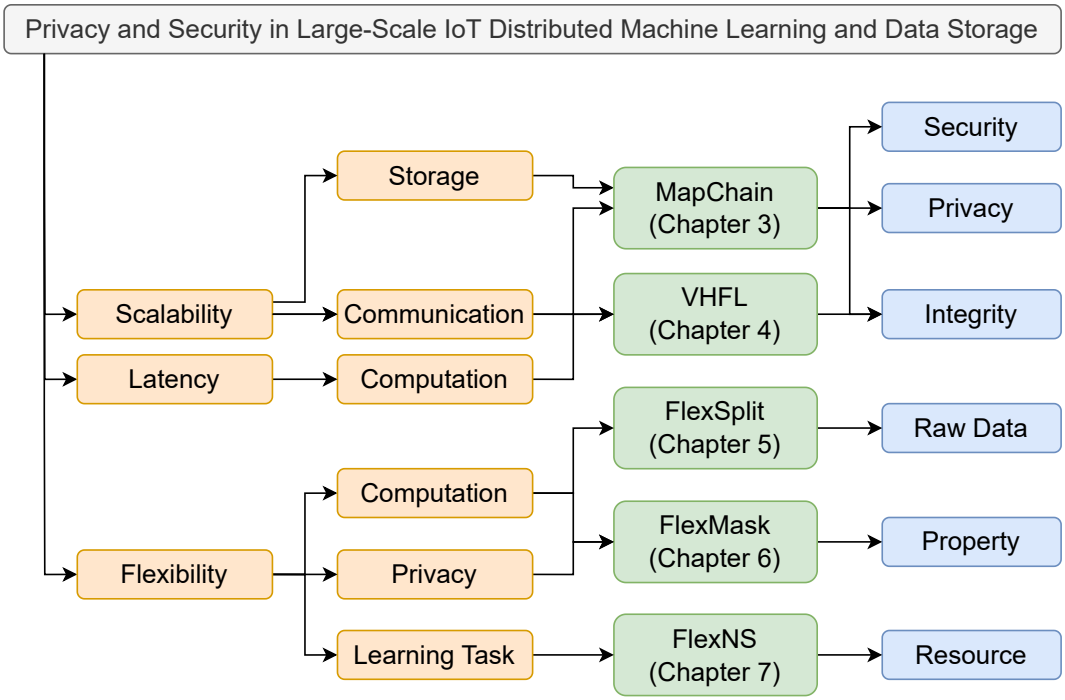


FIGURE 1.2. Thesis Outline

under different resource limitations, as well as privacy and security requirements for each edge client device. The techniques proposed in this thesis are based on state-of-the-art architectures such as blockchain [9], Federated Learning (FL) [10], Split Learning (SL) [11], [12] and Transfer Learning (TL) [13] with privacy and security enhancements. Those techniques are designed to be integrated with large-scale heterogeneous IoT networks and support flexible personalisation for devices with various resource constraints and privacy requirements. In particular, the goal is to improve scalability and reduce latency and resource consumptions in conventional IoT data storage and DML systems with enhancements in data integrity, security and privacy.

The detailed contributions of this thesis are listed as follows:

(1) A dual-blockchain system model named *MapChain-D* is proposed.

- The proposed *MapChain-D* model separates data storage, block generation (also known as "mining") and data collection into three distinct distributed components. The aims of the proposed *MapChain-D* model are to improve scalability and heterogeneity, as well as increase computing efficiency by introducing more parallel processing on multiple devices at different hierarchical layers.
- The data insertion and retrieval protocols for *MapChain-D* are designed to ensure that they are feasible to be integrated with practical IoT platforms. Theoretical analysis of the scalability of *MapChain-D* is performed to highlight its performance advantages compared to conventional single-chain storage in terms of space complexity, time complexity and communication complexity.
- A practical prototype of *MapChain-D* is developed, supporting multiple RPi and Arduino devices. It is implemented by integrating a Kademlia Distributed Hash Table (DHT) [14] and Ethereum blockchain with low-latency Clique Proof-of-Authority (PoA) consensus [15]. Performance benchmarking of *MapChain-D* is presented compared to conventional single-chain Ethereum solutions with local and distributed storage. The impact of the increasing number of storage and blockchain nodes is investigated to demonstrate the scalability and reliability performance of *MapChain-D*. A fundamental design trade-off of increasing the storage network size to reduce the communication and computing costs of data retrieval is highlighted in *MapChain-D*.

(2) A cloud-edge model integrity verification technique named *VHFL* – Verifiable Hierarchical Federated Learning (HFL) is proposed to mitigate modification attacks on the HFL model parameters and client weights.

- The proposed *VHFL* model and signature aggregation processes are designed to have low resource overheads and support any FL model aggregation scheme that uses a weighted linear combination of client models such as FedAVG and FedSGD [10]. The integrity of the lightweight model signatures is guaranteed by a transparent and tamper-proof blockchain and aggregated by a smart contract. The blockchain does not need to store any model parameters to maximise storage and communications efficiency and protect user privacy.

- Detailed algorithms are presented for the *VHFL* model aggregation and integrity verifications at clients, edge servers, and the cloud server. The proposed solution applies the properties of homomorphic hashing [16] such that only the aggregated model parameters and signatures are needed to detect any inconsistencies in the pre-aggregated models from the clients or servers.
- The integrity verification sensitivity of *VHFL* for identifying model parameters and client weight attacks under different hash precision levels and dropout rates are analysed. The execution time overheads of *VHFL* compared to conventional HFL are further evaluated. Attack simulations are performed at different hierarchical layers and highlight that *VHFL* can effectively recover the utility of the HFL models under attacks by removing malicious or unfairly aggregated models.
- An Ethereum-based private blockchain network is deployed on RPis to evaluate the blockchain overheads of *VHFL* in practical scenarios. Results show that the proposed *VHFL* technique effectively mitigates FL attacks at the client, edge, and cloud layers whilst preserving the utility of the cloud model with lower overheads compared to benchmark schemes.

(3) A configurable, privacy-preserving federated-split learning framework named *FlexSplit* is proposed.

- The proposed *FlexSplit* framework allows each client to select the number of layers shared with an edge server for training according to its privacy constraints. It adopts a hierarchical structure where model training is performed across multiple groups of clients and edge servers. The clients can select an edge server to share part of their model training with other clients.
- The split-aggregate algorithm is designed for *FlexSplit*, enabling the clients to control the private information shared within the system and decide the trade-off between privacy, resource consumption, and model utility. Layer-wise aggregation [17] is utilised to enable the cloud server to aggregate the shared models generated by multiple edge servers with different cut layers.

- *FlexSplit*'s model utility is compared to centralised learning, FL, and SL as benchmarks. *FlexSplit* can achieve a similar validation accuracy as the optimal centralised learning approach when a minority of the clients require more privacy preservation. If the split layer is fixed, *FlexSplit* significantly outperforms conventional FL and SL, highlighting the advantage of allowing clients to select their individual privacy levels. The privacy-utility trade-off in *FlexSplit* is also investigated to show how clients could determine their optimal selection of split layer based on different privacy measures and attack scenarios.

(4) A configurable, personalised masking technique named *FlexMask* is proposed for property inference attack protection in SL.

- A PIA scenario is presented in the SL framework. It is inspired by the conventional early exit technique. The attack is demonstrated by selecting 13 different facial attributes in the CelebFaces Attributes (CelebA) dataset [18] as private attributes in gender, emotion, and age classification tasks using the LeNet-5 [19] Convolutional Neural Network (CNN) model.
- Inspired by the masking techniques in distributed learning [20]–[22], *FlexMask* – a novel masking technique is proposed that applies personalised masks to the split layer activation outputs to defend against PIA for their personalised private attributes. The adversarial model, algorithms, mask determination methods and mask application methods are detailed for the proposed *FlexMask* technique.
- Comprehensive experimental analyses are performed. Results show that the proposed masking technique substantially reduces the attack success rate under most of the considered scenarios. Results show the relationship between the attack success rate and the correlations between the neurons' activation outputs and the private attribute, as well as the Mutual Information (MI) between private attributes and the original learning tasks.

(5) A novel collaborative learning framework named Flexible Neuron Selection (*FlexNS*) is proposed for resource-efficient personalised training on the edge.

- The proposed *FlexNS* framework partially transfers centrally-trained model parameters to resource-constrained IoT edge devices for personalised training

on different tasks. *FlexNS* provides the flexibility for IoT devices to select an exit layer based on their resource constraints. A neuron selection algorithm is proposed to select the neurons at the exit layer that are the most relevant to the target task to reduce further the number of training parameters at the IoT devices.

- Detailed algorithms are presented for neuron separation, neuron selection, and the training process of *FlexNS*. Examples of cluster-based calculations of the separation measurement metrics are presented. Theoretical analysis is conducted from the perspectives of computational complexity and communication cost of the proposed *FlexNS* framework.
- Simulation-based experiments are presented on a network attack detection dataset and an image classification dataset to highlight that *FlexNS* achieves 104.3% and 98.4% of the source model's utility with 76.7% reduction in training time and 91.4% reduction in inference time compared to the conventional split learning framework. Compared to the conventional structured pruning technique, *FlexNS* yields a maximum of 30.57% increase in model utility. Simulations on resource-constrained IoT devices are performed to demonstrate that our approach is feasible in realistic IoT scenarios.

1.3 Thesis Outline

In this thesis, distributed data storage and ML frameworks are proposed to leverage state-of-the-art blockchain, FL and SL paradigms with improvements in security, privacy and efficiency. The rest of the chapters are organised as follows:

- In Chapter 2, we start by discussing the background and related work of this thesis.
- In Chapter 3, a secure and efficient blockchain-based distributed data storage system named *MapChain-D*, is presented. *MapChain-D* adopts a dual-blockchain structure that separates data storage, block generation and data collection aiming to improve scalability and computing efficiency in heterogeneous IoT scenarios.

- In Chapter 4, a verification technique named *VHFL* is presented, which leverages homomorphic signatures to enhance the trust and security of HFL model aggregation. In *VHFL*, the lightweight model signatures are stored in a blockchain, allowing all entities at various hierarchical levels to verify models' integrity efficiently.
- In Chapter 5, a hierarchical federated-split learning framework named *FlexSplit* is presented. *FlexSplit* introduces multiple edge servers with various split layers, which enables a personalised privacy level for each client by selecting the edge server with the appropriate split layer.
- In Chapter 6, a personalised masking-based privacy preservation technique named *FlexMask* is presented. In *FlexMask*, a layer in the client-side model is chosen as the mask layer with the most sensitive neurons masked to protect attributes of the clients' private dataset, which might not be relevant to the original learning tasks.
- In Chapter 7, a personalised neuron selection technique named *FlexNS* is presented based on SL, TL and early exit, allowing clients to train a lightweight model with personalised learning tasks using a subset of neurons in an early layer of a larger ANN pre-trained by the cloud server.

For each framework presented in this thesis, detailed algorithms and theoretical analysis are included. To analyse the proposed frameworks, comprehensive simulation- and emulation-based experiments are performed with results presented.

CHAPTER 2

Literature Review and Background

This chapter summarises the background of this thesis based on the literature review, including blockchain-based distributed data storage and different variants of DML paradigms. Security and privacy threats in DML are discussed, and existing security and privacy preservation solutions are presented. Drawbacks and limitations of current solutions are identified as the research gaps to highlight the contributions of this thesis.

2.1 Blockchain-Based Distributed Data Storage

Blockchain is also known as a type of Distributed Ledger Technology (DLT) [9] where the records of the ledger are stored as "transactions" in a chain of "blocks". Compared to traditional databases, the data storage in a blockchain system is fully decentralised and does not require a centralised server [23]. To ensure the data stored in the distributed network is immutable and transparent, the fundamental features, including hash-based block structure, consensus algorithm and decentralised architecture [23], are considered in blockchain systems. As blockchain does not rely on any centralised trusted entity, a block should only be generated after a majority of participants reach a consensus on the transaction states [24]. To improve data security and integrity further, activities such as retrieving and modifying blockchain data are traceable and accountable using smart contracts [25].

Fig. 2.1 illustrates the basic structure of a blockchain with blocks n and $n + 1$. The basic unit of a blockchain is a "block" formed by the "block header", which stores the information of the block, and the payload (also known as "block body") that contains the transactions.

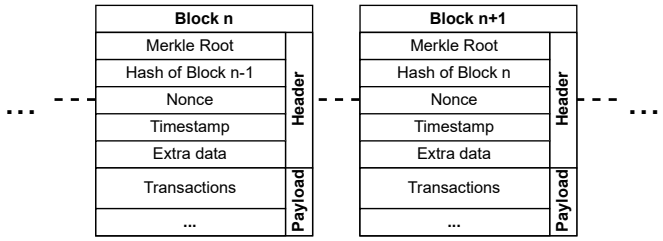


FIGURE 2.1. Blockchain Basic Structure

One of the key components of the block header is the hash of the previous block used to verify the integrity of the blockchain. The hash digestion should include the header and payload of the previous block. Due to the fact that the block header always contains the hash of the previous block (except for the first block), tampering with one of the blocks in the chain results in an avalanche effect, causing the hashes of all the blocks after it to be invalid. Apart from the previous block's hash, the root of the "Merkle Tree" can also be included to efficiently verify the state of the blockchain when the block is generated. A Merkle tree is a binary tree where the leaves are the hashes of each of the previous blocks, where each of the parent nodes is the hash of its child nodes [26].

2.1.1 Challenges in Conventional Blockchain-based Data Storage Systems

Blockchain is a fully decentralised database with a tamper-proof guarantee. It can be regarded as an immutable insert-only P2P distributed data structure where the data cannot be deleted or tampered with after insertion. This makes it a promising solution to manage the trust of distributed data storage in IoT [24], [27]. In conventional blockchains such as Ethereum and Bitcoin, the storage and communication issues have been identified due to the large communication and storage requirements caused by the propagation and storage of large volumes of transactions at each of the edge devices [28], as well as the computation and storage consumption when a new or unstable edge device requests historical data from others when (re)joining the system [29]. Those issues can be overcome by dividing the end devices (i.e., blockchain nodes) into full nodes that store the entire ledger and light nodes that only

store the block headers [30], [31]. However, the light nodes can only verify transactions stored on the full nodes [31], which increases the complexity of data retrieval and insertion when retrieving the raw data.

2.1.2 Peer-to-Peer Data Storage

P2P storage system is a type of distributed data storage system where the data is stored in a decentralised manner on multiple edge devices (also known as "peers"). The "peers" in a P2P network interact with each other directly to propagate the data during data insertion and retrieval operations [32] without needing a centralised server. The storage resources are shared, and decisions are made autonomously in the distributed network [32]. DHT [33], [34] is a type of structured P2P storage system where each of the data objects (values) is stored deterministically at one or more peers in the P2P network with a unique identifier (key) [35]. The data stored in the DHT network is expected to be uniformly distributed based on the hash function used to calculate the key [36]. The uniform distribution and redundancy of the data improve the efficiency of data insertion and retrieval and prevent single-point-of-failure [36].

2.1.3 Blockchain-Based P2P Data Storage

Different from dividing the blockchain nodes into full nodes and light nodes, an alternative approach to reduce the overall storage size of blockchains is to leverage a P2P storage system [31] such as DHT [33], [34]. Conventional P2P storage is not designed to guarantee the trustworthiness and integrity of data. Therefore, compared to the conventional P2P storage systems without a blockchain, integrating P2P storage systems with blockchains can provide an additional layer of integrity and security for the data stored using the blockchain's validation and authentication mechanisms, respectively.

On-chain P2P storage: In [37], [38], the authors proposed to utilise a DHT-based P2P storage system to store the blocks in a blockchain as data objects in a DHT network containing multiple blockchain nodes as peers. However, in heterogeneous IoT networks, different end devices have various resource constraints, making it inefficient to store the entire blockchain

distributively. However, in large-scale heterogeneous IoT networks, it is important to consider the diversity of resources and communication constraints for different edge devices [39]. Therefore, recent works have considered applying hierarchical or multi-layer structured blockchains for IoT data storage to maximise efficiency in heterogeneous IoT networks. In [40], a cloud-edge hierarchical blockchain structure was proposed to improve the efficiency of data storage in heterogeneous IoT networks where historical data is offloaded to the cloud storage instead of occupying the edge nodes' storage space. A double-layer blockchain network proposed by Fan et al. [41] separates the blockchain storage network into a consensus layer and storage layer which are responsible for block generation and storage correspondingly. Xu et al. [42] proposed a two-layer blockchain-based multi-cloud storage system that divides the IoT nodes into different roles, and the IoT nodes in different roles maintain two different ledgers. Alternatively, Liao et al. [43] proposed a graph-based blockchain structure where the edge nodes can choose to partially store the blockchain or store the entire chain based on their resource constraint and latency requirements.

Off-chain P2P storage: Apart from the on-chain P2P storage, it is also possible to consider off-chain P2P storage of the raw data [44], while a blockchain is adopted as a data management system [45]. For example, the authors in [45], [46] proposed to store the raw data as data objects in an off-chain DHT network, and a blockchain is used to store the keys to reference the data objects containing the raw data. The blockchain-based distributed storage proposed by Li et al. [25] utilises blockchain as a trusted third party to manage the data stored in a DHT by keeping records of the addresses of data and clients' identities in blockchain "transactions". The blockchain can provide verification and authentication during data insertion and retrieval processes to enforce the integrity and security of the stored data, respectively [25]. Similar off-chain data storage was developed in [31], [47].

2.2 Distributed Machine Learning

DML is a promising paradigm where the clients perform ML model training locally using the data they collect and send their locally-trained model to the server for further processing.

This is widely used in IoT systems, allowing the clients to keep their data private and only share the trained model to benefit other clients in the system. Instead of sharing raw data samples, the clients only share model parameters trained locally in parallel, reducing the communication costs and privacy leakage and improving the computational efficiency.

The global model utility of DML is expected to be similar to the conventional centralised learning approach due to the same volume of data used for training. It is expected that the global model utility outperforms the local model utility due to the fact that a single IoT client holds a limited amount of data samples compared to the data collection of the entire IoT network. There are some key advantages of DML listed as follows:

- It offloads the computation from the cloud server to all the clients in the network, which reduces single-point-of-failure, as well as the dominant power of the cloud service provider.
- Data privacy is protected because only the trained models are transferred within the network and shared with a centralised entity instead of the raw client data.
- The volume of data transferred in the network is reduced because the edge devices only transfer the model parameters trained using raw data collected within a time period to the cloud server. This saves the network bandwidth and reduces private information shared within the network.

In this thesis, the following variants of DML – FL and SL are leveraged.

2.2.1 Federated Learning

FL is a type of widely-used DML approach proposed by McMahan et al. [10], where all the participating client devices train a shared model collaboratively with the coordination of a centralised cloud server. As illustrated in Fig. 2.2, FL performs collaborative ML model training locally at participating devices (i.e., clients) without the need to share their private training data in the network [10]. The cloud server in FL is only responsible for aggregating the trained model shared by all clients [10].

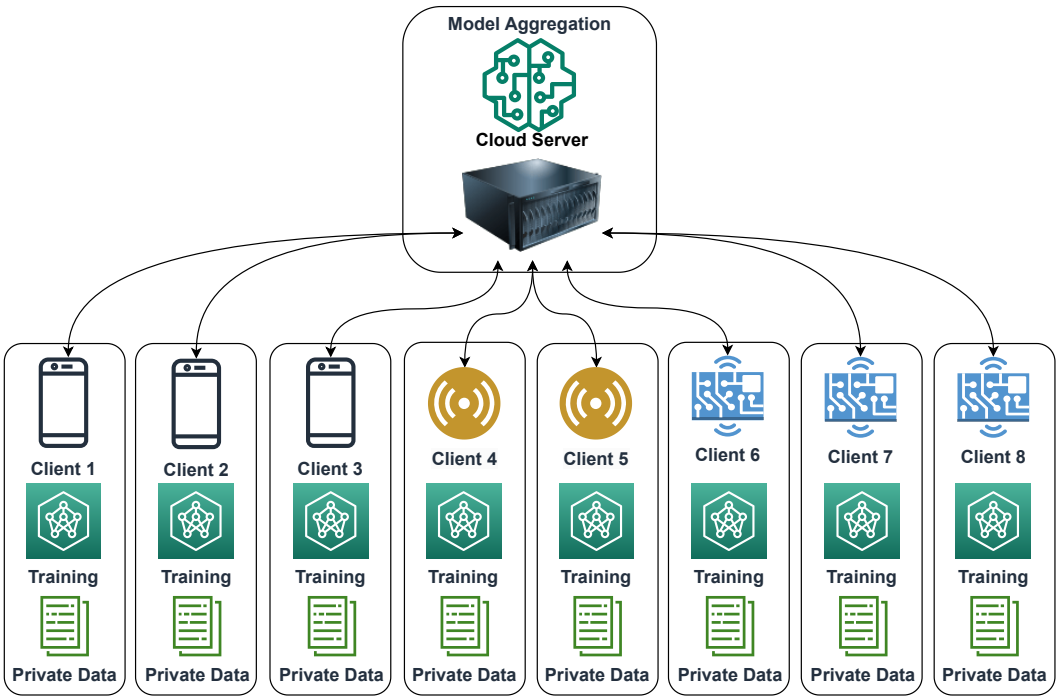


FIGURE 2.2. An Example of the Federated Learning Structure

The conventional FedAvg and FedSGD methods [10] performed on the server aggregate the model weights or gradients using the average values from all the individual client model updates sent to the server following the procedures below:

1–Client local training: In each FL training round, every client performs local training and sends their trained model parameters (i.e., model weights or gradients) $\mathbf{w}_m$ to the server for aggregation.

2–Client weights determination: Let N be the number of clients in the system, and n_m be the size of the local dataset at a given client m. Next, the server determines the client weight applied to each client's model parameters, which is defined in Eq. (2.1):

$$e_m = \frac{n_m}{\sum_{k=1}^{N} n_k} \tag{2.1}$$

3–Averaging model parameters: After determining the client weights, the server calculates the aggregated model parameters $\mathbf{w}$ using Eq. (2.2):

$$\mathbf{w} \leftarrow \sum_{k=1}^{N} \left(e_k \times \mathbf{w}_k\right). \tag{2.2}$$

4–Parameter updating: After performing model aggregation, the server multicasts the aggregated model parameters $\mathbf{w}$ to all the clients. The clients update their local model parameters before proceeding to the next training round.

2.2.2 Hierarchical Federated Learning

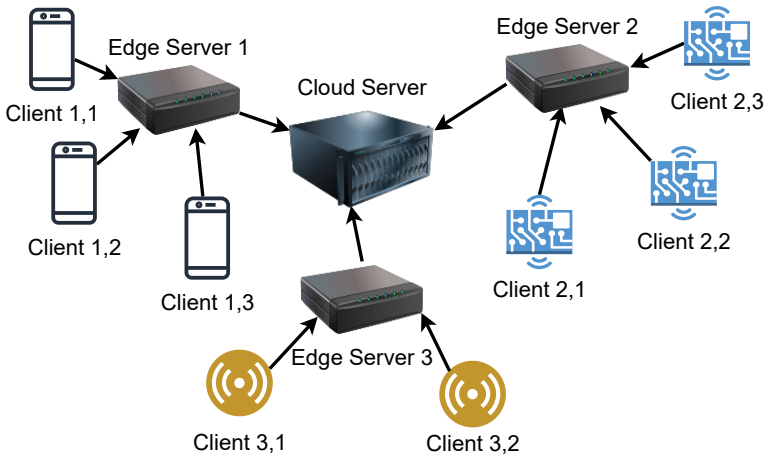


FIGURE 2.3. An Example of the Hierarchical Network Topology with End Devices, Edge Servers and the Cloud Server

One of the key issues of the conventional FL is that it adopts a star topology similar to Fig.1.1 where all participants are connected to a centralised cloud server. This creates a many-to-one network traffic bottleneck and single point of failure, which hinders the deployment of FL in large-scale systems. In order to address the scalability issue caused by the conventional FL's star topology, a hierarchical topology similar to Fig.2.3 could be adopted. The hierarchical topology introduces intermediate "edge servers" that serve as "local centres" and connect multiple local sensor nodes to the cloud server. The computing load at the remote cloud server can be offloaded to the edge server near the clients. Since the number of edge servers is only a fraction of the number of clients, the number of devices connected to the cloud server is reduced significantly, especially for large-scale systems.

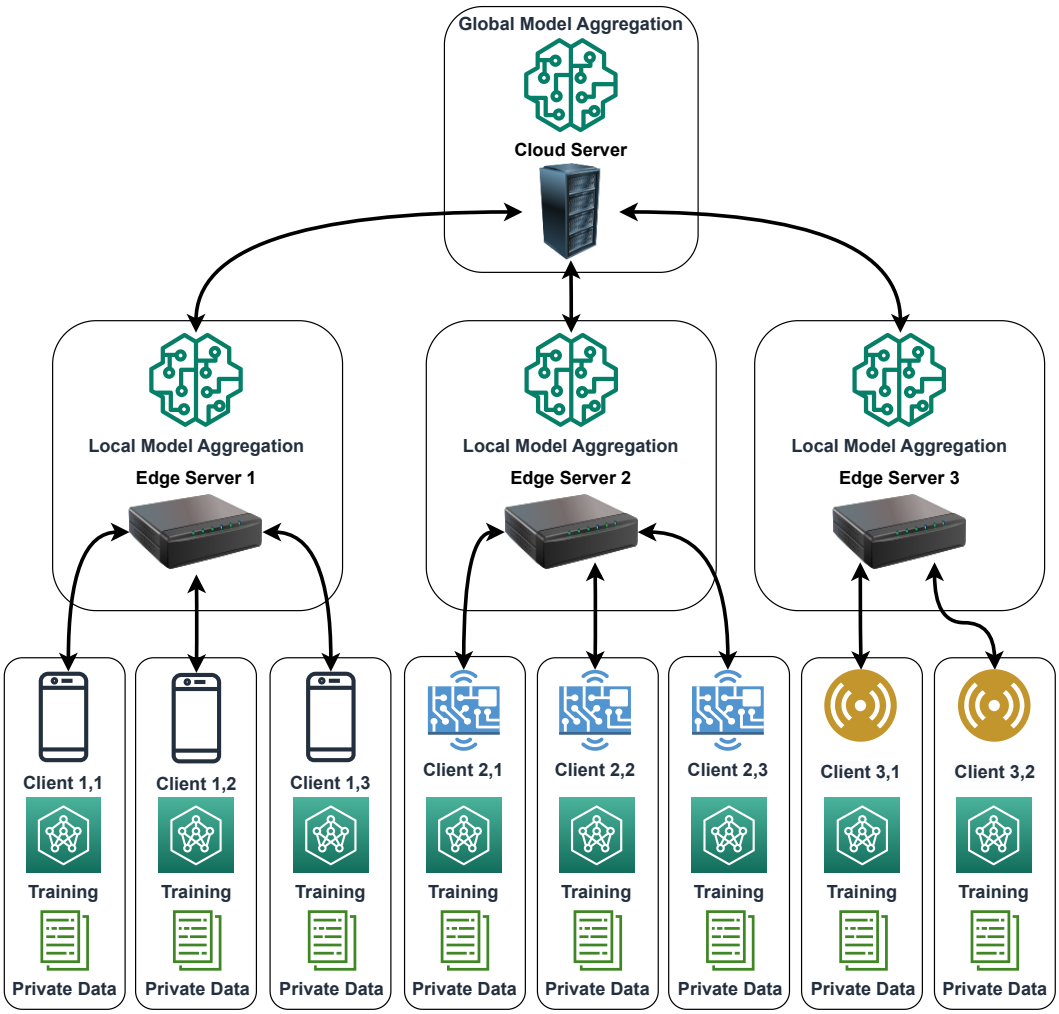


FIGURE 2.4. An Example of the Hierarchical Federated Learning Structure

HFL [48]–[52] is proposed based on this hierarchical topology. As shown in Fig. 2.4, each of the clients is connected to one of the nearby edge servers to upload their locally-trained model. The edge servers are responsible for performing partial aggregation (i.e., local model aggregation), which aggregates all the clients' models from the corresponding "edge group" before sending the locally aggregated model to the cloud server for global model aggregation. The three-level HFL in Fig. 2.4 can be further extended to topologies with multiple levels of hierarchy, or flexible structures where the clients can connect to different edge servers at various training rounds.

HFL offloads model aggregation tasks to multiple edge servers near the clients and reduces the bandwidth required to transfer all client models to the remote cloud server. Therefore, the

scalability is improved compared to conventional FL. Experiments in [50], [53] show that HFL reduces the model training time and communication overheads compared to the conventional cloud-based FL because of the parallel computation and communication processes. It also helps to improve the overall model utility for practical scenarios with hierarchical statistical heterogeneity [54]. Additionally, HFL promotes client-specific security where each client can select their own trusted edge group [55] to prevent reconstruction attacks from model gradients [56]. Differential privacy techniques can also be integrated with HFL to reduce privacy leakage in HFL [57], [58]. However, since the model aggregation is performed by multiple devices belonging to various entities, privacy and security threats are still present in HFL [59].

2.2.3 Split Learning

As the model size increases, performing local training on resource-constrained client devices becomes less feasible. To address the issue, SL [11], [12] is proposed as a type of semi-decentralised DML where an ANN model is split into multiple parts at split layers. The cloud server with stronger computing power performs training of some intermediate layers of the ANN, whereas the clients train layers closer to the input and output layers locally to preserve their privacy.

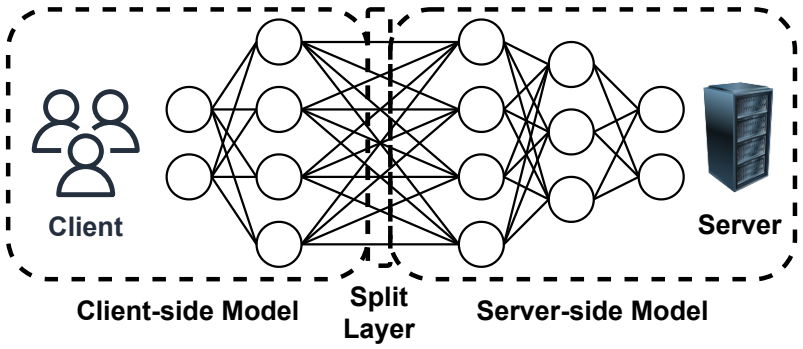


FIGURE 2.5. Overview of Split Learning

Similar to FL, SL model training does not require the clients to transmit their private datasets to the centralised server. In SL, only the split layer activation outputs and gradients are required

to be transferred between clients and the server for forward and backward preparations, accordingly. Compared to FL, SL requires less data to be transferred to the cloud server because FL requires the parameters of the entire model to be transferred between clients and the server after completing a training round. The simplest version of SL is illustrated in Fig. 2.5, where there is only one split layer between clients and the server. In this setup, the clients are required to share their output labels with the server for model training.

2.3 Security in Distributed Machine Learning

The promising DML paradigms introduced in the previous Section move the entire or part of the model training to the edge at client devices. Similar to a typical distributed system, this comes with trust and security issues due to the model training process being controlled by multiple entities in the system. This Section presents security threats against model integrity and privacy in DML systems, as well as model verification and privacy preservation techniques.

2.3.1 Integrity Attacks in Distributed Machine Learning

In a conventional DML network, all clients are assumed to be honest which contributes positively to model training and maximises the global model utility. However, this assumption is unlikely to hold in practical scenarios because the clients are independent and not controlled by any external party. Therefore, a client can provide unmeaningful model parameter updates or train the model maliciously [60] to reduce the global model utility or lead the model to behave according to their interests.

Data poisoning attacks: Cao et al. [61] proposed a distributed poisoning attack in FL networks, where multiple devices perform training using a malicious dataset. Data poisoning attacks are performed before the model training phase, where only the data used for training is poisoned, and the training process is uncompromised. In order to increase the attack performance further, Zhang et al. [62] proposed the use of a Generative Adversarial Network (GAN) to generate samples to poison the model training.

Model poisoning attacks: Instead of manipulating the training data before the model training phase, an attacker can also poison model parameters after training to reduce model utility [60] or add a backdoor to the model [63]. The model poisoning attacks can be classified into untargeted attacks and targeted attacks based on the capability of the attackers. Different from data poisoning attacks, the attackers in model poisoning attacks can gain more control over the behaviour of the target model since they have full control of the model training process [60]. Additionally, the attack can be performed on various devices, such as the clients, servers or the communication links between them. Therefore, this type of attack is considered stronger than data poisoning attacks [60], [63].

Model weight attacks: As shown in Eq. (2.2) in Sec. 2.2.1, the FedAvg and FedSGD methods aggregate model parameters by calculating the weighted linear combination of the individual client model parameters. The weights applied to different client models may depend on the client's dataset size, reputation, and trust level [10], [64], [65]. Therefore, instead of manipulating the model parameters or training datasets in FL, an attacker can also change the weight of the client models such that the model aggregation is biased towards the models of interest [66], [67]. Model weight attacks are more difficult to detect since the clients' datasets are private to the individual clients. Therefore, no external party is able to determine the correct weights of the clients' models due to the limitations of the information one can obtain in the system [66].

2.3.2 Model Verification Techniques in Distributed Machine Learning

Model verification techniques are important in DML as the clients rely on the server to aggregate models without knowing all individual models. A third party might also need to audit the model aggregation processes to ensure their correctness. The aggregated model needs to be verified without any knowledge of the individual client model parameters.

Cryptography-based model verification: The conventional DML model verification is to utilise cryptographic functions to generate key pairs for verification. For example, the authors in [68] combined bilinear pairing, Paillier encryption and Chinese remainder theorem to verify

model aggregation results in FL. They assumed that the hash function and private key used for signature calculation are unknown to the aggregation server. However, the aggregation server could collude with any of the clients to obtain the hash function and private key, and generate valid signatures for compromised models. To overcome such colluding issues, Fu et al. [69] proposed a privacy-preserving FL model verification scheme to verify the correctness of model aggregation using Lagrange interpolation. Their proposed approach requires a trusted third party to initialise the model parameters, generate keys, and create a pseudo-random number generator, increasing the complexity of implementation and introducing additional communication costs and a single point of failure.

Homomorphic hashing [16], [70] is a promising approach that can be leveraged in DML model aggregation verification because the model aggregation results can be verified using the hashes of individual models without knowing the individual model parameters. Model verification using homomorphic hashing does not require complicated key exchange and cryptography algorithms (e.g., encryption and decryption). Guo et al. [71] proposed to utilise the homomorphic hashing algorithm to verify the integrity of the model aggregation. Xu et al. [72] proposed to utilise the homomorphic hashing algorithm with the pseudo-random technique to verify the integrity of the FL model aggregation with failure tolerance. Their framework also utilised double masking based on the Diffie-Hellman key agreement to protect clients' privacy [72]. However, this approach has scalability issues, as identified by the authors in [55]

To enhance its trust and security, model verification processes can be performed in a Trusted Execution Environment (TEE). For example, Zhang et al. [73] proposed utilising the TEE to perform model verification in FL aiming to mitigate model modification and free-rider attacks. The scheme they proposed also aims to protect the privacy of the clients' private training data [73].

Blockchain-based model verification: An early approach of fully decentralised DML proposed in [74] leverages a permissioned blockchain to perform model aggregation in a fully decentralised manner. However, the confidentiality of the models in the blockchain is highly dependent on the access control scheme adopted by the blockchain. The trustworthiness

of the model aggregation is not guaranteed in this conventional blockchain-based model aggregation approach. Similar fully decentralised blockchain-based FL architectures are proposed in [75]–[78],

To mitigate trust issues of blockchain-based DML, Kim et al. [79] proposed an alternative approach to store all individual client model parameter updates into a blockchain. Miners perform cross-verification before inserting the model parameters to ensure the correctness of the individual client models. After inserting the model parameters, the clients retrieve all individual model parameters and perform model aggregation locally [79]. Similar on-chain model aggregation techniques are proposed in [80]–[83] with additional TEE integration for secure local training [82], hierarchical structure to improve scalability [81] or consensus mechanism [81], [83]. These approaches obviously incur scalability and privacy issues because the unprocessed client model parameters are uploaded to the blockchain and accessible by all the clients in the system. They also consume large storage and communication resources, especially for larger models and systems with more clients.

Instead of storing all client model parameters, the authors in [84], [85] proposed only storing the proofs for the models in the blockchain to guarantee the integrity of the FL models. The authors in [86]–[89] proposed to utilise blockchain-based incentive mechanisms to encourage honest parties in the system to improve fairness [86]–[88] or defend against malicious participants [89] during collaborative model training. The authors in [90], [91] proposed to utilise blockchain to provide auditability of the FL model training and aggregation. Ouyang et al. [92] further generalised a blockchain-based framework for distributed collaborative learning, assuming that all the participants could be untrusted. The trust in blockchain-based FL can also be improved by using a reputation mechanism [93], [94]. Furthermore, Shayan et al. [95] integrated blockchain with differential privacy, Multi-Krum robust aggregation and secure aggregation to protect the privacy, fairness and integrity of decentralised FL systems.

Byzantine-robust aggregation: Byzantine-robust aggregation protocols are a group of statistical-based model aggregation protocols in DML. The goal of Byzantine-robust aggregation is to defend against model poisoning attacks by removing the statistical outliers from the model aggregation [96]. Krum [97] is a commonly used Byzantine-robust aggregation

algorithm based on the Euclidean distance between the gradients of models. Only the models with gradients closer to their neighbours are selected, whereas the models that have gradients far away from their neighbours are treated as outliers to be removed from the aggregation [97]. Alternatively, Yin et al. [98] proposed calculating the median or trimmed-mean (i.e., mean value without k greatest and smallest values) of the gradients to remove the outliers in model gradients. The authors in [99] proposed to combine [97], [98]. Cao et al. [64] proposed a solution where the server trains a model using a clean dataset and computes the cosine similarity between its gradients and the gradients received from the clients. The gradients with higher cosine similarity are regarded as more trusted. To deal with the heterogeneity of the models trained by non-Independent and Identically Distributed (IID) datasets (e.g., data with different geographical, time or user features), Li et al. [100] proposed to group the FL clients and perform Byzantine-robust aggregation within each of the groups.

2.4 Privacy in Distributed Machine Learning

Apart from the security concerns in DML, which mainly focus on model integrity, privacy issues in DML have recently been explored. One of the key advantages of DML compared to conventional centralised learning is that the clients' private data is not shared in the system. However, recent research [56], [101], [102] showed that the clients' private information, such as the raw input data, output labels, or properties of the data in FL could be reconstructed using the information embedded in the model parameters shared in the system. Geiping et al. [103] further extended those privacy attacks to reconstruct higher-resolution image data. The authors in [104], [105] further generalised the model reconstruction attacks in FL. This Section presents different types of privacy attacks according to adversaries' knowledge and capability, as well as the adversarial targets. Finally, some commonly used privacy preservation techniques will be presented.

2.4.1 Adversaries' Knowledge and Capability

According to the adversaries' knowledge and capability, privacy attacks in DML can be divided into white-box attacks, black-box attacks and query-free attacks [106], [107]. In black-box attacks, the model is treated as a black box, where the adversaries do not have any knowledge about the model, but are able to inject any data into the model, execute inference queries and receive the corresponding outputs from the model [106]. It is also possible for a black-box adversary to have some knowledge (e.g., distributions, sample values) of the training data [106]. Comparatively, in white-box attacks, the model structure and parameters are known to the adversaries, but the adversaries cannot execute inference queries directly from the original model [106]. Query-free attacks do not require the model parameters to be exposed to the adversaries, and the adversaries are unable to execute any inference query from the original model [106]. The adversaries in query-free attacks only know the distribution of the training data with auxiliary information such as model structure and some values of the training samples, whereas none of the model parameters and input/output pairs is known to the adversaries [106].

2.4.2 Adversarial Targets

The adversarial target (i.e., the information that the adversary wishes to reveal) of the privacy attacks in DML can be classified into data reconstruction attacks, PIA and MIA [108]. In a data reconstruction attack, the adversaries aim to reconstruct the original training data (raw data) by eavesdropping on the model parameter updates and performing model inversion algorithms [107], [109], [110], or attaching a GAN to the shared model parameters [111], [112]. The attacks considered in [109], [110] are black-box attacks where the adversaries are able to obtain some pairs of input data and activation outputs from the split layer of the SL client-side model, but they have no knowledge of the client-side model. The private data could be reconstructed using any split-layer activation outputs. Comparatively, the adversaries in [107] only need to know the structure of the model, which is regarded as an enhanced version of the white-box attack. In particular, the adversaries in [107], [110] construct a

"shadow model" using the known server-side model to mimic the original client-side model's behaviour.

In PIA, the adversarial target is the property of the training data. The property can be the original learning objective (i.e., classification labels) [113]–[117] or hidden features (i.e., properties that are unrelated to the original learning objective) [102], [107], [110]. For example, in a gender classification model with facial images as the inputs, the adversaries can infer the gender or some facial attributes of the input images that are unrelated to the genders [102]. PIAs can be performed during the prediction stage in vertical FL [118], where the adversaries are able to infer the feature using only the prediction outputs from the model.

MIA is a type of attack where adversaries intend to identify if a data sample or an attribute (e.g., clients' identity) appears in the training set. For example, in a natural language model, whether a word appears in a batch of training samples can be distinguished by whether the gradient of the corresponding entry in the word embedding matrix is zero [102]. Similar to PIA, the adversaries can perform MIA by training a shadow model that behaves similarly to the genuine client model, and attaching a binary classifier to classify the presence of the data sample or attribute [119], [120].

2.4.3 Cryptographic-Based Privacy Preservation Techniques in Distributed Machine Learning

Some commonly used cryptographic-based approaches to prevent privacy attacks in DML are listed below [121].

Homomorphic encryption: Homomorphic encryption is a type of encryption technique that supports additive and multiplicative operations [121]. Paillier encryption [122] is a commonly used homomorphic encryption algorithm. In the homomorphic encryption scheme, the operations can be applied to the encrypted data without needing to decrypt it [123]. The result after applying the operation to the encrypted data remains encrypted and can be decrypted using the same key [123]. This makes it suitable for use in DML because the clients can share the encrypted model parameters with the server and the server can

perform model aggregation with model parameters remaining encrypted [124]. Ma et al. [125] utilised a multi-key homomorphic encryption scheme to encrypt models from different clients using various keys, which further enhances the models' privacy for different clients in the system. The decryption of the aggregated model requires a key aggregation, which aggregates all public keys of the individual client models [125]. Zhang et al. [126] improved the computation and communication efficiency of homomorphic encryption in FL by adopting a batch encryption mechanism. The authors in [127], [128] improved the computational efficiency in homomorphic encryption-based privacy-preserving DML schemes.

Secure aggregation: Secure aggregation is a type of DML model aggregation approach based on secure multi-party computation [129]. One of the earliest approaches to secure aggregation was proposed by Bonawitz et al. [130], where the clients leverage a Diffie-Hellman key exchange protocol to encrypt the shared model parameters using the double-masking mechanism. The double-masking mechanism applies pairwise masking between clients' model parameters before transmitting them to the server for model aggregation. The model aggregation results are decrypted if the aggregation is performed correctly at the server. Build on top of [130], Choi et al. [131] proposed to select a subset of client pairs to perform secret sharing, aiming to reduce communication and computation costs. So et al. [132] proposed a secure aggregation scheme with more tolerance to client dropout and reduced communication overheads. Li et al. [133] proposed a chain-based architecture for secure model aggregation. Some recent works have improved communication costs, robustness against attacks, client dropouts and system heterogeneity in secure aggregation [134]–[137].

Masking-Based privacy preservation techniques in DML: Sparse learning is a commonly used technique in DML to reduce communication costs by only updating a selective subset of the training parameters [138]. It can also be used as a technique to reduce the amount of information obtained by a third party during communication [138]. In the previous paragraph, double-masking was discussed as a secure aggregation scheme in DML where all the clients need to apply masks in pairs or in a chain, and the aggregated model is decrypted by cancelling out all the masks applied to the individual client models. However, the cryptographical-based masking schemes require complicated key exchange among all pairs of clients in the system,

making it inefficient, especially for large-scale heterogeneous IoT systems. Additionally, they might not be robust to colluding clients [139]. Apart from masking the input data [140], Ergun et al. [22] proposed applying masks to the shared model parameters and leveraging the secure aggregation technique to reduce privacy leakage during client model-sharing processes in FL. The privacy-efficiency trade-off of the masking technique has also been discussed in [22].

In SL model training, only the split layer activation outputs and gradients are shared in the network. Vepakomma et al. [109] proposed utilising the distance correlation minimisation technique applied to the input data and the split layer activation outputs in order to defend against model inversion attacks in SL. Li et al. [141] proposed adding bottleneck layers at the split layer and inversion models to defend against model inversion attacks. Alternatively, Mao et al. [142] proposed to replace the deterministic activation function with a probability-based activation function to reduce privacy leakage in SL.

2.4.4 Differential Privacy-Based Privacy Preservation Techniques in Distributed Machine Learning

Differential Privacy (DP)-based DML privacy preservation scheme [143] is a type of non-cryptographical approach that adds an artificial noise [144]–[146], shuffles the model parameters [147], [148] or performs dropout [140] during communications, aiming to protect the clients' raw data or labels. Different from cryptographical approaches, the cost of DP-based privacy preservation is the decrease in model utility due to the reduction of accuracy of the information shared in the system. Adaptive approach of DP methods such as [149], [150] considering the trade-off between privacy leakage and model utility. Compared to the conventional DP, the global model utility of the adaptive DP can be improved by achieving an optimised balance between privacy and performance loss.

2.4.5 Hybrid Privacy Preservation Techniques in Distributed Machine Learning

It is also possible to combine multiple approaches above to strengthen the privacy-preservation performance. For example, the authors in [151]–[153] proposed hybrid approaches that considered combining multiple methods to protect clients' privacy. Xu et al. [154] proposed a hybrid approach using secure aggregation, DP and functional encryption, aiming to reduce communication costs while guaranteeing privacy.

2.5 Resource Reduction Techniques in Distributed Machine Learning

This Section introduces three techniques to reduce resource consumption in DML: TL, which transfers model parameters from a well-trained model to another model that performs different tasks; Model pruning, which compresses the full model by reducing the number of nodes (neurons) and edges (connections) in an ANN; Early exit which exits an ANN model at an early layer.

2.5.1 Transfer Learning

TL transfers the learned knowledge via model parameters from a pre-trained model (i.e., source model) to a new model (i.e., target model) that performs a different task [13]. Compared to the conventional approaches that train the model from scratch, TL significantly reduces training time and resource consumption by leveraging the pre-trained source model [155], [156]. The model pre-training is usually performed on servers with more computation capabilities, whereas the target model will be fine-tuned by the resource-constrained edge devices using their local data according to their local training tasks [156]. Typically, the target model should follow the same structure as the source model to allow the transfer of the parameters of the full model. A common approach to fine-tune the target model is to freeze the early layers (i.e., layers closer to the input layer) of the model and re-train the later

layers (i.e., layers closer to the output layer) [157], [158] or fine-tune the parameters in those layers [159]

2.5.2 Model Pruning

Han et al. [160] proposed a model compression technique, known as model pruning, that removes redundant parameters in an ANN model to reduce model complexity. Model pruning also helps accelerate model training and inference, especially for edge devices with limited computational and communication resources [161], [162]. Model pruning can be categorised into structured pruning and unstructured pruning. In structured pruning, the model structure is changed by removing filters or channels within one or more layers, or removing one or more full layers of the ANN model [163]. Comparatively, the conventional unstructured pruning [160] removes the connections between neurons (i.e., edges), and the original ANN model structure remains unchanged.

2.5.3 Early Exit

Early exit [164], [165] is a technique proposed to exit an ANN classification model before the output layer by connecting a simple Single-Layer Perception (SLP) classifier with a single fully connected layer between an early exit layer and the output labels. Layer-wise training [166], [167] is proposed by leveraging the early exit technique that connects an auxiliary SLP classifier to each intermediate shallower sub-models sequentially to train the full ANN model layer-by-layer. The early exit and layer-wise training approaches allow a larger model to be trained under limited resources. Early exit also helps reduce inference time, especially in resource-constrained IoT scenarios [168], [169]. In order to reduce the number of connections between the exit layer and the SLP classifier, a max-average pooling [170] or attention [171] module can be added between the exit layer and the SLP classifier.

CHAPTER 3

Scalable and Secure Distributed Data Storage and Communications

---

3.1 Introduction

With billions of IoT devices [1] continuously generating huge amounts of sensor and actuator data, the conventional approach of sending all data to remote cloud data centres becomes nonviable as it will congest the network, cripple service delivery performance and hinder future growth. Moving computation, storage and analytics to the edge devices closer to the end-user has shown great potential to address such performance challenges [172]. However, those IoT edge devices are often deployed in uncontrolled public networks with minimal trust and security compared to the cloud data centres. Additionally, different groups of edge devices may have conflicting interests and divergent trust requirements.

As a result, ensuring trust and security becomes an important requirement for distributed IoT data management for the end-user to extract useful information for data fusion and mining tasks [173]. Blockchain is a promising paradigm for IoT distributed trust management due to its tamper-proof, decentralised and transparent natures in maintaining a distributed ledger. Additionally, activities such as data retrieval and modification are traceable using smart contracts deployed on top of a blockchain to enforce data integrity further [25]. The conventional approach of blockchain-based data storage typically assumes that all edge devices have sufficient storage, computation and communication resources capable of operating a blockchain [174]. However, this assumption is unlikely to be valid in practically large-scale IoT networks where the edge devices have heterogeneous resource constraints.

To improve the practicability of deployment of blockchain-based data storage systems under heterogeneous resource constraints, the *MapChain-D* framework is proposed in this chapter, which adopts a dual-blockchain structure storing the raw data and managing their indexing information using two mapped blockchains. The two blockchains are stored and managed by edge devices with various levels of resource constraints and trust levels. In particular, the raw data samples are stored at a blockchain-based P2P storage network consisting of selected trusted devices with larger storage capacity, higher computational power and more bandwidth. Meanwhile, the indexing information is stored in a lightweight blockchain propagated within the entire network. The proposed dual-blockchain system model improves the efficiency, trust and security of the data by considering the heterogeneity of large-scale IoT systems. The inherited immutable property of blockchains guarantees the integrity of the raw data. Detailed algorithms for data insertion and retrieval protocols, as well as theoretical and experimental analysis, are presented for the proposed *MapChain-D* framework.

3.2 System Model

3.2.1 MapChain-D System Model and Entity Groups

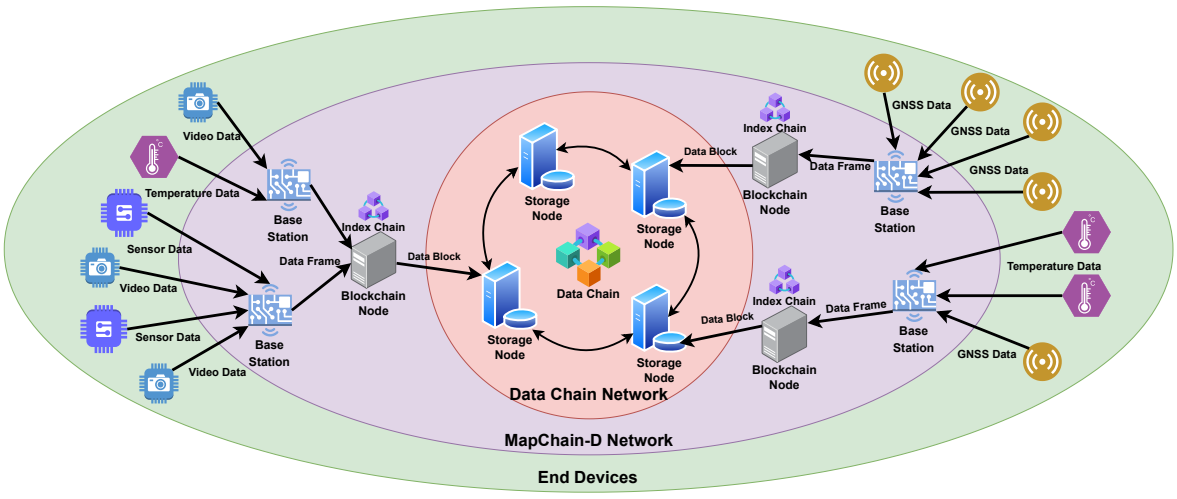


FIGURE 3.1. System Overview of the Proposed MapChain-D Framework

Fig. 3.1 shows an overview of the main components of the proposed *MapChain-D* system model, which is a distributed IoT data storage framework aimed at maximising the efficiency, privacy and security of IoT data by storing different types of information in different entity groups. *MapChain-D* framework contains three hierarchical layers of the devices performing different tasks in the system.

Each entity group consists of devices with similar hardware resources (e.g. computing power, storage capacity, network conditions and energy consumption) and trust levels. The devices in the same entity group perform the same role. Data collection, data management and data storage operations are performed by devices in different entity groups to maximise efficiency since different stages of data processing can be performed by the devices in different entity groups in parallel.

To maximise the storage efficiency and security of the IoT data, the IoT data should be maintained by a smaller number of devices belonging to entity groups with relatively higher trust levels and more available hardware resources. On the other hand, the references to the data objects are maintained by a larger number of devices and transparent to all entity groups in the system to maximise lookup efficiency and data integrity. The entity groups are detailed in the following:

End-device: End-devices are the IoT nodes that sense the surrounding environment and transmit their captured data to the base stations. Each of the end devices in the *MapChain-D* system is connected to and managed by at least one of the base stations in the *MapChain-D* network. The end-devices are registered and authenticated to the communications network which allows them to connect securely to various base stations at different times. It is assumed that the end-devices have the strictest resource constraint and are infeasible to perform complex onboard data processing and storage.

Base station: Base stations provide the communications interface for the end-devices to transmit data to the *MapChain-D* network. It is assumed that each of the base stations is managed by a data provider where all the end-devices connected to it are managed by the same data provider to protect the integrity and confidentiality of the data. Each base

station should be connected to one of the blockchain nodes via a local network to insert data into the data chain network. The base stations are also responsible for sorting the data and removing unusable data (i.e., corrupted or duplicated data frames) from the edge-devices before they are sent to the blockchain nodes. It is assumed that the base stations have relatively more computing and communication resources than the end-devices and are capable of key exchange, device authentication and data processing functionalities.

Blockchain node: Blockchain nodes are the most trusted and reliable devices with minimal resource constraints. Blockchain nodes are also known as "miners" where a block generation process is executed by an authenticated blockchain node when a certain amount of data frames have been received from the base stations. Blockchain nodes are also responsible for processing data retrieval requests from the authorised consumers or auditors. Therefore, they are also responsible for device authentication during data retrieval operations. A blockchain node should connect to one or more storage nodes via a local network for modification and retrieval of the "state" of the blockchain. If a blockchain node connects to more than one storage node, one of them acts as the main storage node, whereas the rest are for backup if the main storage node fails.

Storage node: All the storage nodes in the *MapChain-D* framework form a P2P data storage network. The storage nodes are connected according to a mesh or fully-connected network topology to propagate data objects during data insertion and retrieval operations. The storage nodes located at different sites are assumed to be connected via secure encrypted Internet links. Storage nodes are managed by the storage service providers where a storage service provider could maintain multiple storage nodes at different sites to contribute more storage space, balance the load and maximise the efficiency. To ensure the integrity and confidentiality of the data, the storage nodes should only accept data insertion and retrieval requests from authenticated blockchain nodes.

Consumer: The consumers, or end-users in the *MapChain-D* IoT framework request data from one of the blockchain nodes. Lookup operations would then be performed within the P2P storage network formed by the storage nodes to retrieve the data objects of the requested data

block. The consumers could be managed by various entities and their system configurations could be different.

3.2.2 Dual-Blockchain Structure and Mapping Scheme

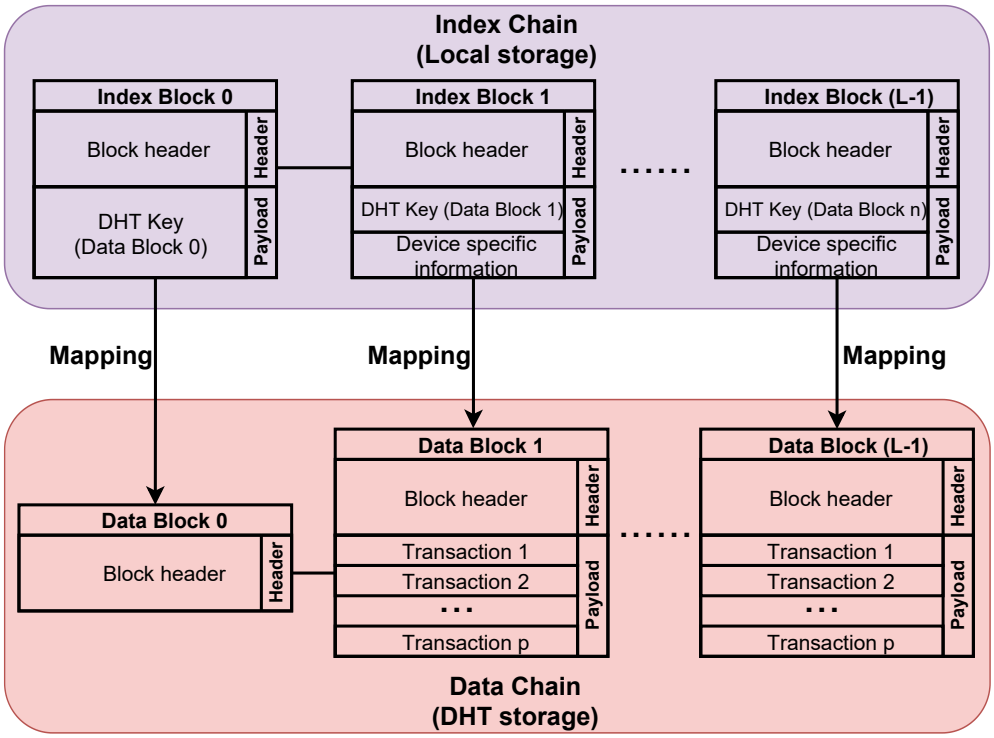


FIGURE 3.2. *MapChain-D* Dual-Blockchain Mapping Structure

MapChain-D framework proposes separating the data storage and indexing components into two mapped blockchains that are maintained by entity groups with similar hardware resources and trust levels in the *MapChain-D* network. As shown in Fig. 3.2, the blockchain structure of *MapChain-D* consists of two chains: a data chain storing the IoT data in a DHT-based P2P data structure and an index chain storing indexing and verification information for the IoT data. The IoT data is stored in the data field of the transactions in each data block, except for block 0 which is the genesis block that contains no transaction and is created when a *MapChain-D* network is initialised. A one-to-one mapping structure is considered where each of the index blocks is mapped to a data block using the DHT key of the data block. The one-to-one mapping mechanism reduces the implementation complexity, where the DHT keys can be directly used to link the two chains.

Note that *MapChain-D* is developed as a distributed framework for the dual-blockchain DHT mapping architecture where the base stations are only responsible for communication, whilst the block generation and block storage functions are performed in parallel at separate blockchain and storage nodes, respectively. This improves the feasibility of implementing *MapChain-D* in practical heterogeneous scenarios and reduces the consequences of single-points-of-failures.

3.2.3 Data Integrity and Confidentiality

Due to the fact that the storage location of the data is randomised and the data is stored in a DHT-based blockchain, the difficulty of tampering with the data is relatively high. The proposed *MapChain-D* data exchange protocol includes a validation procedure that ensures malicious tampering can be detected quickly and the transparency of the index chain further increases the difficulty of tampering with the data without notification, ensuring that the data integrity can be validated by all consumers.

The advantage of data integrity can be improved by introducing more data providers or storage service providers to increase the level of decentralisation of the system such that the data storage is not dominated by a centralised entity. The data providers which manage groups of end-devices are responsible for maintaining the integrity of their own data. Distributed solutions for trust management can be built based on this, where no centralised entity is required to enforce the trust level of the data. Various authentication and trust management schemes can be integrated with *MapChain-D* for data providers with different trust levels.

Furthermore, the index chain can store encrypted DHT keys that can only be decrypted by the storage service providers (e.g. storage nodes) so that the exact position of each individual data block is unknown by the blockchain nodes to avoid targeted attacks. It is assumed that none of the information stored in the index chain is confidential where the index chain can only be used by the consumers to query and validate the data from the data providers. The data providers are responsible for ensuring the authenticity of the requesting consumer to enforce the data confidentiality.

Finally, it should be noted that only a random partition of the storage nodes are responsible for the propagation of a data block in *MapChain-D* model and there can be multiple propagation paths. This reduces the consequence of single-point-of-failure and attacks since the DHT network provides redundancy for data storage and propagation.

3.2.4 Consensus Algorithm

As the conventional Proof-of-Work (PoW) permissionless consensus algorithm in the Ethereum blockchains requires a large amount of computing power for the "miners" to generate blocks [174], [175] which is infeasible for many IoT applications [29], Byzantine Fault Tolerant (BFT) mechanisms have been developed as an alternative to the PoW consensus mechanism to provide sufficient fault tolerance [176]. For permissioned blockchains, PoA is a type of consensus mechanism based on BFT with less message exchange and better performance [176]. In the PoA consensus, only a selected group of highly-trusted nodes (also known as "validators") are able to generate a block in the blockchain [177].

By adopting PoA consensus in *MapChain-D*, the computing power requirements for block generation are reduced compared to PoW consensus. Additionally, the integrity of the data is enforced since the blocks can only be inserted by a limited number of trusted nodes.

3.3 *MapChain-D* Protocols

This section details the proposed data exchange protocols of *MapChain-D*.

3.3.1 Data Insertion

For data insertion, a blockchain node in *MapChain-D* model needs to communicate with one of the storage nodes to insert a data block and the data object would then be propagated and stored within the data chain network. The states of the blockchain are intrinsically shared in the DHT network and synchronisation between the blockchain nodes is not required. If a new blockchain node joins the system, it can directly connect to the nearest storage node and access

the data without synchronising the entire data chain from the storage node. Furthermore, only a random partition of the storage nodes are responsible for the propagation of data blocks in *MapChain-D* model. This provides more capability to handle multiple operations simultaneously and reduces the one-to-many network traffic which could potentially create a bottleneck during a block insertion operation.

Algorithm 1 describes the data insertion operation which is triggered when a data frame $\mathcal{M}$ is received by a blockchain node from an end-device $\mathcal{D}$ via a base station. $\mathcal{S}$ is defined as the device-specific information which contains some basic descriptions of the data frame (e.g., location, data provider, etc.) for data lookup. Multiple insertion operations could be performed at different blockchain nodes in parallel. If multiple insertion requests arrive at a single blockchain node simultaneously, the requests from an end-device with a higher reputation will be prioritised.

Note that only `getDataBlock`, `insertDataBlock` and `insertIndexBlock` functions in this algorithm are executed within the *MapChain-D* network where `insertDataBlock` incurs a standard PoA consensus process such that the data block can only be inserted by a validator. If the blockchain node is not a validator, the proposed block will be sent to a validator to be validated and inserted into the data chain when there is no currently ongoing block generation operation. Otherwise, the block generation will be queued. Therefore, $BlockchainLocked$ would be a semaphore shared by all the blockchain nodes. For convenience, the semaphore $BlockchainLocked$ can be stored as a special data object in the data chain network where all the blockchain nodes are able to retrieve the exact same state of the semaphore. If there is a conflict where two blocks are generated simultaneously, the order of the blocks will be on a random basis.

Due to an index block should only be generated and inserted after the corresponding data block is successfully generated and inserted (i.e., Lines 19-22 in Algorithm 1), the index chain follows the same PoA consensus as the data chain and the order of the blocks in both chains should be the same to ensure the one-to-one mapping relationship between two chains. `Hash` is the hash function which computes the hash value for block validation, as well as the

DHT key of a data block for data lookup. Parameter p_{max} can be tuned to ensure the number of blocks in the awaiting generation queue is not constantly increasing.

Algorithm 1 `Insert`$(\mathcal{M}, \mathcal{D}, \mathcal{S})$

Initialisation:

- Private key of the blockchain node: PK
- Index chain: $\mathcal{C}_I$
- Data chain: $\mathcal{C}_D$
- Pending transactions: P
- Block size: p_{max}

Input:

- Data frame: $\mathcal{M}$
- End-device ID: $\mathcal{D}$
- Device-specific information: $\mathcal{S}$

$T = \texttt{sign}(PK, \{\mathcal{D}, \mathcal{M}\})$ ▷ Initialise transaction
$P[\texttt{length}(P)] \leftarrow T$ ▷ Add to pending transactions
if $\texttt{length}(P) > p_{max}$ **then** ▷ Commit pending transactions
  if $BlockchainLocked == True$ **then**
    Wait until $BlockchainLocked == False$
  end if
  $BlockchainLocked = True$
  $l \leftarrow \texttt{length}(\mathcal{C}_I)$
  $\mathcal{B}_{I\_prev} \leftarrow \mathcal{C}_I[l-1]$ ▷ Previous index block
  $\mathcal{B}_{D\_prev} \leftarrow \mathcal{C}_D.\texttt{getDataBlock}(\mathcal{B}_{I\_prev}.H_D)$ ▷ Previous data block
  $H_{D\_prev} \leftarrow \texttt{Hash}(\mathcal{B}_{D\_prev})$
  $\mathcal{B}_{D\_new} \leftarrow \{l, t, H_{D\_prev}, P\}$ ▷ New data block
  $t \leftarrow \texttt{getTimeStamp()}$
  $H_{I\_prev} \leftarrow \texttt{Hash}(\mathcal{B}_{I\_prev})$
  $H_D \leftarrow \texttt{Hash}(\mathcal{B}_{D\_new})$
  $\mathcal{C}_D.\texttt{insertDataBlock}(\mathcal{B}_{D\_new})$
  if $DataBlockInserted$ **then**
    $\mathcal{B}_{I\_new} \leftarrow \{l, t, H_{I\_prev}, H_D, \mathcal{S}\}$ ▷ New index block
    $\mathcal{C}_I.\texttt{insertIndexBlock}(\mathcal{B}_{I\_new})$
  end if
  $P \leftarrow \{\}$ ▷ Reset pending transactions
  $BlockchainLocked = False$
end if

3.3.2 Data Retrieval

Algorithm 2 is the function for a data retrieval operation. As an example, the search target is the latest data frame where the timestamp is less than t. Therefore, linear lookup is performed starting from the latest index block and returns the first data frame which matches the search criteria (i.e., timestamp less than t). If no result matches, the function will return the oldest data frame in the chain. Note that `getDataBlock` is the only function that requires interaction with the data chain network. Similar to the data insertion operations, multiple data retrieval operations could be executed in parallel from multiple blockchain nodes. Each of the blockchain nodes is responsible for maintaining a queue to execute the retrieval requests sequentially based on their arrival time.

`checkUserAuth` is a function that checks the authenticity of the data consumer to access the requested data. Using this function, the requested data frame will only be returned if the data consumer has permission to access the data. Otherwise, *UserUnauthenticatedError* will be raised (See Lines 23-27 in Algorithm 2). Apart from *UserUnauthenticatedError*, *InvalidIndexBlockError* or *InvalidDataBlockError* will be raised if there is a failure to validate the index block or data block (i.e., the calculated hash value for the previous block is not equal to the value stored in the current block), correspondingly.

3.3.3 Search Requirements

As an example, the insertion and retrieval operations in Algorithms 1 and 2 consider a simple timestamp-based search where the data consumers demand the latest data frame created before a particular timestamp. However, it should be noted that the algorithms can be adjusted based on the searching tasks and properties of the data recorded, and the device-specific information (i.e., $\mathcal{S}$) can be utilised to perform a more precise search before interacting with the data chain network. Additional information might be recorded as device-specific information in the index chain for more efficient local search in the blockchain nodes and to minimise the communication costs caused by the propagation of the data objects within the data chain network.

Algorithm 2 `Retrieve(t)`

Initialisation:

- Index chain: $\mathcal{C}_I$
- Data chain: $\mathcal{C}_D$

Input:

- Timestamp: t

Output: Data frame: $\mathcal{M}$

$i \leftarrow \texttt{length}(\mathcal{C}_I) - 1$
$i_{target} \leftarrow 1$
while $i > 1$ **do** ▷ Linear lookup for index block
  if $\mathcal{C}_I[i].t < t$ **then**
    $i_{target} \leftarrow i$
    exit loop
  end if
  $i \leftarrow i - 1$
end while
$\mathcal{B}_{I\_target} \leftarrow \mathcal{C}_I[i_{target}]$
$\mathcal{B}_{I\_prev} \leftarrow \mathcal{C}_I[i_{target} - 1]$
if $\mathcal{B}_{I\_target}.H_{I\_prev} != Hash(\mathcal{B}_{I\_prev})$ **then**
  raise *InvalidIndexBlockError* ▷ Index block invalid
end if
$H_{D\_prev} \leftarrow \mathcal{B}_{I\_prev}.H_D$
$H_{D\_target} \leftarrow \mathcal{B}_{I\_target}.H_D$
$\mathcal{B}_{D\_prev} \leftarrow \mathcal{C}_D.\texttt{getDataBlock}(H_{D\_prev})$
$\mathcal{B}_{D\_target} \leftarrow \mathcal{C}_D.\texttt{getDataBlock}(H_{D\_target})$
if ($\mathcal{B}_{D\_target}.H_{D\_prev} == \texttt{Hash}(\mathcal{B}_{D\_prev})$ and
$H_{D\_target} == \texttt{Hash}(\mathcal{B}_{D\_target})$) **then** ▷ Data block is valid
  if `checkUserAuth()` ==True **then**
    return $\mathcal{M}$
  else
    raise *UserUnauthenticatedError*
  end if
else
  raise *InvalidDataBlockError*
end if

Besides, the example considers the order of the data insertion as indexes which is suitable for timestamp-based search. The optimal indexes could be different for different search criteria (e.g., temperature, location, sensor reading) and it is important to select the index based on the properties of the data and search criteria to optimise the efficiency of the data retrieval operations. Additionally, for data of larger size (e.g., video data), the payload should be

divided into multiple data frames to avoid extensive delay and communication usage while creating and propagating a data block. Dividing the data into smaller data frames makes the storage and communication loads more balanced and improves the efficiency during data insertion and retrieval operations. The maximum size of a data frame needs to be defined such that any data frame exceeding the maximum size will be divided into multiple data frames of at most the maximum size before executing Algorithm 1.

3.3.4 Privacy Measures

The privacy measures of the *MapChain-D* are inherited from the blockchain privacy measures. For example, zero-knowledge proof techniques and access control schemes can be integrated with the blockchain data storage network. The multi-layer structure of *MapChain-D* ensures that the original data is stored at a limited number of trusted devices (i.e., storage nodes) within a trusted network environment (i.e., data chain network). Comparatively, the majority of the client devices in the system have limited access to the original data. All of the entities in the system can verify data using the information stored in the tamper-proofed index chain without accessing the data chain, ensuring the verifiability and confidentiality of the original data.

3.4 Theoretical Complexity Analysis

This section details theoretical complexity analysis from the perspectives of space complexity, time complexity and communication complexity.

The storage space complexity of the *MapChain-D* framework and the time complexity of the data exchange protocols are analysed. Theoretical comparisons are presented with the comparison between conventional single-chain frameworks with local and distributed data storage. The complexity analysis is summarised in Table 3.1. In the analysis, The constants adopted in the analysis are listed as follows:

- Number of storage nodes N_{Ψ};

- Number of blockchain nodes N_{Φ};
- Length of index and data chains (i.e., number of blocks) L;
- Storage size of index chain $S_{\mathcal{C}_I}$;
- Storage size of data chain $S_{\mathcal{C}_D}$;
- Storage size of index block $S_{\mathcal{B}_I}$;
- Storage size of data block $S_{\mathcal{B}_D}$.

TABLE 3.1. Complexity Analysis for *MapChain-D* Compared to Conventional Single-Chain Local and Distributed Storage Systems

Complexity		Single-Chain Local Storage	Single-Chain Distributed Storage	***MapChain-D*** Blockchain node	***MapChain-D*** Storage Node
Space		$O\left(S_{\mathcal{C}_D}\right)$	$O\left(\frac{S_{\mathcal{C}_D}}{N_\Phi}\right)$	$O\left(S_{\mathcal{C}_I}\right)$	$O\left(\frac{S_{\mathcal{C}_D}}{N_\Psi}\right)$
Insertion	Time	$O\left(S_{\mathcal{B}_D} \times \log N_\Phi\right)$	$O\left(S_{\mathcal{B}_D} \times \log N_\Phi\right)$	$O\left(S_{\mathcal{B}_I} \times \log N_\Phi\right)$	$O\left(S_{\mathcal{B}_D} \times \log N_\Psi\right)$
	Commun.	$O\left(S_{\mathcal{B}_D} \times N_\Phi\right)$	$O\left(S_{\mathcal{B}_D} \times N_\Phi\right)$	$O\left(S_{\mathcal{B}_I} \times N_\Phi\right)$	$O\left(S_{\mathcal{B}_D} \times \log N_\Psi\right)$
Retrieval	Time	$O\left(L\right)$	$O\left(L\right) \times O\left(\log N_\Phi\right)$	$O\left(L\right)$	$O\left(\log N_\Psi\right) + O\left(\frac{L}{N_\Psi}\right)$
	Commun.	0	$O\left(L\right) \times O\left(\log N_\Phi\right)$	O(1)	$O\left(\log N_\Psi\right)$
Summary (Assumed $N_\Phi = N_\Psi$)		High space complexity intermediate insertion and low retrieval complexity	Low space complexity intermediate insertion and high retrieval complexity	Low space complexity intermediate insertion and intermediate retrieval complexity	

Number of storage nodes N_Ψ; Number of blockchain nodes N_Φ; Length of index and data chains (number of blocks) L; Index chain storage size $S_{\mathcal{C}_I}$; Data chain storage size $S_{\mathcal{C}_D}$; Index block storage size $S_{\mathcal{B}_I}$; Data block storage size $S_{\mathcal{B}_D}$

Note: $S_{\mathcal{C}_I}$ is significantly smaller than $S_{\mathcal{C}_D}$, $S_{\mathcal{B}_I}$ is significantly smaller than $S_{\mathcal{B}_D}$.

Note that each index block has the same storage size, whereas the data blocks could be of various sizes depending on the amount of data traffic in the network at a given time. If we assume that the number of nodes in the single-chain frameworks is equal to the number of blockchain nodes in *MapChain-D*, the storage size of the blockchain in the single-chain frameworks is equal to the size of the data chain, and the storage size of a block in the single-chain frameworks is equal to the size of a data block in *MapChain-D*.

3.4.1 Space Complexity

In *MapChain-D*, the storage nodes are responsible for storing a uniformly randomised part of the data chain according to the DHT structure. Therefore, the space complexity of each storage node is:

$$O\left(\frac{S_{\mathcal{C}_D}}{N_\Psi}\right) \tag{3.1}$$

Furthermore, a full copy of the index chain is stored at each of the blockchain nodes. Therefore, the space complexity of each blockchain node is:

$$O\left(S_{\mathcal{C}_I}\right) \tag{3.2}$$

In single-chain models with local storage, the full copy of the blockchain is stored at each of the nodes. Therefore, the space complexity of each node in single-chain models with local storage is:

$$O\left(S_{\mathcal{C}_D}\right) \tag{3.3}$$

If distributed storage is utilised in the single-chain models, the space complexity is reduced to:

$$O\left(\frac{S_{\mathcal{C}_D}}{N_\Phi}\right) \tag{3.4}$$

Since the storage size of the index chain $S_{\mathcal{C}_I}$ is significantly smaller compared to that of the data chain $S_{\mathcal{C}_D}$, the *MapChain-D* model has a significantly smaller storage size complexity than the single-chain model with local storage, i.e., for each storage node:

$$O\left(S_{\mathcal{C}_D}\right) > O\left(\frac{S_{\mathcal{C}_D}}{N_\Psi}\right) \tag{3.5}$$

and for each blockchain node:

$$O\left(S_{\mathcal{C}_D}\right) > O\left(S_{\mathcal{C}_I}\right) \tag{3.6}$$

Furthermore, the storage space consumption of *MapChain-D* reduces with an increasing number of storage nodes, N_Ψ, in the network. Similarly, for single-chain frameworks with distributed data storage, the storage space consumption reduces with an increasing number of blockchain nodes, N_Φ, in the network. In summary, *MapChain-D* achieves similar space complexity compared to the single-chain model with distributed storage and lower than single-chain models with local storage.

3.4.2 Time and Communication Complexity of Data Insertion

For the *MapChain-D* insertion operation, the time complexity for a blockchain node to generate an insertion request is:

$$O\left(S_{\mathcal{B}_D} \times \log N_\Psi\right) \tag{3.7}$$

since the latest data block needs to be retrieved from the DHT-based data chain to compute H_{D_prev}. Utilising the commonly used gossip protocols, the new index block can be propagated to all the blockchain nodes with the time complexity of:

$$O\left(S_{\mathcal{B}_I} \times \log N_\Phi\right) \tag{3.8}$$

and communication complexity of:

$$O\left(S_{\mathcal{B}_I} \times N_\Phi\right) \tag{3.9}$$

The time and communication complexities for the DHT-based storage node lookup and the propagation of a new data block are:

$$O\left(S_{\mathcal{B}_D} \times \log N_\Psi\right) \tag{3.10}$$

Therefore, for *MapChain-D*, the overall time complexity of the data insertion operation is:

$$O\left(S_{\mathcal{B}_I} \times \log N_\Phi\right) + O\left(S_{\mathcal{B}_D} \times \log N_\Psi\right) \tag{3.11}$$

and the overall communication complexity is:

$$O\left(S_{\mathcal{B}_I} \times N_\Phi\right) + O\left(S_{\mathcal{B}_D} \times \log N_\Psi\right) \tag{3.12}$$

For the single-chain model with local storage, each newly generated block needs to be propagated to all the blockchain nodes for synchronisation so that each of the blockchain nodes maintains the exact same copy of the blockchain. Therefore, the overall time complexity of the insertion operation in a conventional single-chain with local storage is:

$$O\left(S_{\mathcal{B}_D} \times \log N_\Phi\right) \tag{3.13}$$

and the overall communication complexity is:

$$O\left(S_{\mathcal{B}_D} \times N_\Phi\right) \tag{3.14}$$

For an insertion operation of single-chain models with distributed storage, the overall time complexity is:

$$O\left(S_{\mathcal{B}_D} \times \log N_\Phi\right) \tag{3.15}$$

and the overall communication complexity is:

$$O\left(S_{\mathcal{B}_D} \times N_\Phi\right) \tag{3.16}$$

However, the actual values are expected to be slightly higher than those with local storage, as the previous block needs to be retrieved from the distributed storage network at the time complexity of:

$$O\left(S_{\mathcal{B}_D} \times \log N_\Phi\right) \tag{3.17}$$

and communication complexity of:

$$O\left(1\right) \tag{3.18}$$

We see that the overall time and communication complexities of data insertion in *MapChain-D* are similar to single-chain models with distributed storage and slightly higher than single-chain with local storage due to the insertion of the index block.

3.4.3 Time and Communication Complexities of Data Retrieval

For the *MapChain-D* data retrieval protocol, the linear lookup for an index block is first executed at a blockchain node with the time complexity of $O(L)$. Furthermore, a query with the requested index will be sent to the DHT-based data chain network followed by a lookup for the storage node and the requested data block with average time complexities of:

$$O(\log N_{\Psi}) \tag{3.19}$$

and

$$O\left(\frac{L}{N_{\Psi}}\right) \tag{3.20}$$

respectively [36]. It is assumed that if the data block is found in the network, it will be returned to the requesting blockchain node directly at a constant time. Note that the local lookup for an index block is performed in parallel at multiple blockchain nodes, whereas the lookup for the data block is performed sequentially within the data chain network. The communication complexity for sending the request and receiving the response to the nearest storage node is:

$$O(1) \tag{3.21}$$

During the execution of the retrieval of a data block, the communication rounds required for the lookup of the storage node that stores the requested data block is:

$$O(\log N_{\Psi}) \tag{3.22}$$

If the targeted storage node is found, the data will be directly returned to the blockchain node with constant communication complexity. For single-chain models with local storage, only a linear lookup is required with the time complexity of:

$$O(L) \tag{3.23}$$

Multiple lookups could be executed in parallel at each node. Of course, no communication is needed for *MapChain-D* data retrieval since each node has a local copy of the entire chain. Comparatively, single-chain with distributed data storage has the least efficient data lookup at

both time and communication complexities of:

$$O(L) \times O(\log N_{\Phi}) \tag{3.24}$$

as the linear lookup is performed within the distributed data storage. We see that the overall time and communication complexities of data retrieval in *MapChain-D* are higher than single-chain models with local storage and lower than single-chain models with distributed storage.

3.5 Experimental Prototype

In this section, the experimental prototype of our *MapChain-D* framework is introduced, which is designed according to the main components described in Section 3.2.1.

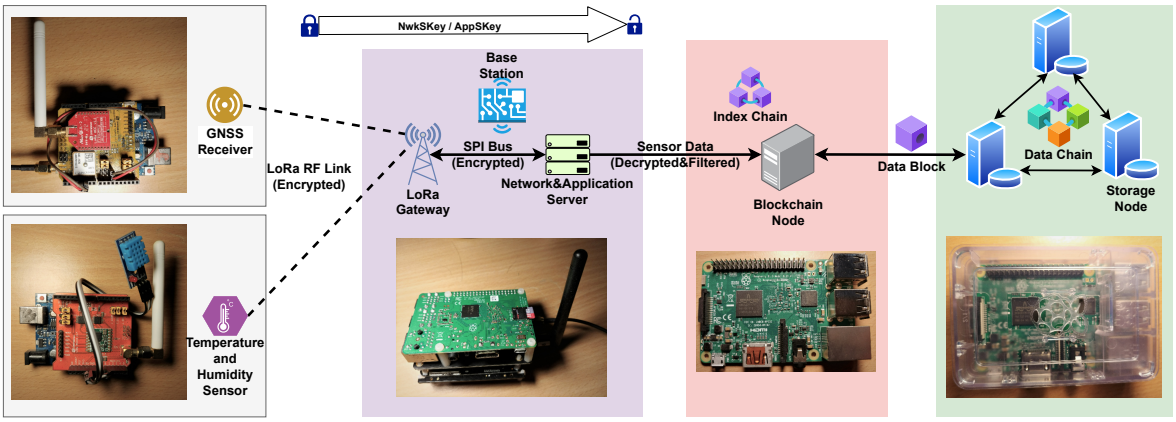


FIGURE 3.3. *MapChain-D* Testbed Configuration and Experimental Hardware

3.5.1 *MapChain-D* Testbed Configuration

Fig. 3.3 illustrates an overview of the testbed configuration deployed and the hardware used for the experiment. In the integrated *MapChain-D* framework, a distributed Long-Range Wide Area Network (LoRaWAN) system architecture is considered with multiple IoT data providers managing their own end-devices, Long-Range (LoRa) gateways, network servers and application servers. This is different from a traditional centralised LoRaWAN network

topology [178] where the gateways are connected to a single central network entity and the application servers are managed by a third-party [179].

In summary, all the devices used in the testbed are listed in Table 3.2.

TABLE 3.2. *MapChain-D* Testbed Device List

Role	Device and Peripherals
Blockchain Node	RPi 3 Model B
Storage Nodes	12 x RPi 3 Model B
Network Server & LoRa Gateway	RPi 3 Model B+ with RAK831 LoRa Gateway Module
End-Device Temperature & Humidity Sensor	2 x Arduino UNO with DHT11 Sensor & Dragino LoRa Shield
End-Device GNSS Receiver	2 x Arduino UNO with Dragino LoRa GPS Shield
Network Switch	Netgear GS316

3.5.2 Testbed Requirements and Configuration

End-device: In the testbed, the end-devices support LoRa Radio Frequency (RF) technology, and follow the LoRaWAN activation procedure and communication protocol. Arduino UNO based LoRa transceivers are used as end-devices for the experiment since it is low-cost and open-source. Various types of sensor nodes are attached to the end-devices to provide different types of sensor data for experimental purposes. As an example, DHT11 temperature and humidity sensors and Dragino GNSS receivers are used. Advanced Encryption Standard (AES)-128 algorithm is used for encrypting the LoRa data frames.

Base station: The LoRa gateways receive data frames from the end-devices via LoRa RF link and transmit them to the network server. For the hardware configuration, the base station contains a RAK831 LoRa gateway module connected to an RPi 3 Model B+ using a Serial Peripheral Interface (SPI) bus. Since the experiment is conducted in the same location and all the sensor nodes are within the range of one gateway, a single gateway configuration is implemented. Multiple gateways could be easily connected to the network server for extended

wireless coverage. ChirpStack Gateway Operating System (OS) [1] is installed on the RPi and it is configured to be a combined gateway, network server and application server according to the testbed configuration.

Blockchain node: In the testbed, the blockchain nodes are equivalent to Ethereum full nodes. An RPi 3 Model B with Raspbian OS installed is used as the blockchain node. The *MapChain-D* application is implemented using Python3 and the py-evm [2] library modified with the Clique PoA as the consensus protocol to avoid excessive energy and computing power at the blockchain node. The number of virtual machines (i.e., blockchain nodes) in py-evm is set to be equal to the number of storage nodes in the *MapChain-D* simulation. For the benchmark comparison, single-chain Ethereum blockchains are implemented using py-evm with Clique PoA consensus protocol. psutil [3] library is installed for accessing the system utilisation for performance measurements. The memory usage, Central Processing Unit (CPU) load and network usage presented in Section 3.6 are measured using the psutil library.

Storage node: In the testbed, 12 storage nodes are deployed. All the storage nodes in the testbed form a Kademlia P2P data storage network and are interconnected via an Ethernet switch. Note that the storage nodes could also be connected via the Internet using secure connections such as a Virtual Private Network (VPN). Each storage node is installed with Rasbian OS and has a static IP address. Python3 is installed on all the storage nodes with the Kademlia library [4]. A Transmission Control Protocol (TCP) server is implemented on each storage node which is connected to one of the other storage nodes and listens to any insertion, retrieval and removal requests from the blockchain node. Similar to the blockchain node, psutil is installed on all the storage nodes to monitor the system resource utilisation. Similar to *MapChain-D*, a Kademlia storage network is set up for the single-chain Ethereum with distributed storage.

[1] https://www.chirpstack.io/
[2] https://py-evm.readthedocs.io/en/latest/
[3] https://psutil.readthedocs.io/en/latest/
[4] https://github.com/bmuller/kademlia

3.5.3 Blockchain Insertion and Retrieval Procedures

An insertion operation is executed when a data frame from an end-device sensor node arrives at the LoRa gateway. The raw data after filtering and decrypting by the network and application servers would then be transmitted to the blockchain node using HTTP POST request in JavaScript Object Notation (JSON) format. The blockchain node which receives the data frame generates a data block and an index block if the number of transactions reaches a threshold. Then the new blocks are propagated in the *MapChain-D* network. Each of the blockchain nodes also has a TCP server listening to a port for data retrieval queries. When a consumer requests a data frame, a data retrieval query would be transmitted to a blockchain node. The blockchain node would then perform a data retrieval operation. In the testbed, a blockchain node generates the data retrieval queries automatically for testing purposes. For *MapChain-D* and single-chain Ethereum with distributed storage, the database in Ethereum is replaced with a Kademlia DHT database, and an Application Programming Interface (API) is designed with a simple TCP-based protocol for the communication between the blockchain nodes and storage nodes using JSON format.

3.6 Experimental Results

In this section, experimental results of the proposed *MapChain-D* prototype, and benchmark comparisons with a conventional single-chain Ethereum system, are presented.

3.6.1 Comparison of *MapChain-D* and Ethereum

First, the performance of the data insertion and retrieval protocols for *MapChain-D* is compared with Ethereum with local and distributed data storage. The *MapChain-D* and Ethereum with distributed data storage store raw data frames from the base station in a blockchain-based distributed storage system. For *MapChain-D*, the indexes of the data are stored in a separate chain, whereas Ethereum stores the raw data frames in a single blockchain. The configurations of the DHT-based distributed storage are the same for *MapChain-D* storage nodes and

Ethereum with distributed storage where the data is stored in the memory of the blockchain nodes. For local storage, the blockchain state data is stored in local hash tables in the memory of the blockchain nodes.

To obtain the performance results, 1000 blocks are inserted into the blockchain and blocks are retrieved from various positions within the chain. The time difference between a data block being successfully inserted and the corresponding data frame being received from the LoRaWAN application server is recorded as the insertion latency. The time difference between the completion of retrieval of a data frame and the generation of the corresponding retrieval query is recorded as the retrieval latency. For each retrieval operation, the average retrieval latency across four retrieval queries demanding the latest data frame from each of the four sensor nodes starting at various block positions are used. 1-minute and 5-minute average CPU loads, network and memory usages of the blockchain node during each insertion and retrieval operation are also recorded to assess the computation and communication performance. For a fair comparison, the number of blockchain and storage nodes in the *MapChain-D* network should be the same as the number of blockchain nodes in the Ethereum network. The default k-bucket size for *MapChain-D* DHT storage network is set to 12.

TABLE 3.3. *MapChain-D* Blockchain Node Performance

(A) During Data Insertion

BC Model	Storage Model	Memory Usage (MB)	CPU Load (%1-min)	CPU Load (%5-min)	Network Usage (KB)	Latency (s)
MapChain-D	Distributed	9.47	8.40	8.02	81.72	3.74
Ethereum	Local	11.71	11.10	10.05	13.98	1.10
Ethereum	Distributed	5.78	7.58	7.22	85.68	3.71

(B) During Data Retrieval

BC Model	Storage Model	Memory Usage (MiB)	CPU Load (%1-min)	CPU Load (%5-min)	Network Usage (KiB)	Latency (ms)
MapChain-D	Distributed	11.08	4.64	7.01	13.98	20.33
Ethereum	Local	14.36	15.13	13.15	3.85	4.50
Ethereum	Distributed	8.76	3.41	5.35	42.40	93.94

Table 3.3a shows that during the insertion operations, the memory usage and CPU load are lower for *MapChain-D* and Ethereum blockchain with distributed data storage compared to Ethereum with local data storage. The memory usage is reduced by the efficient DHT distributed storage compared to the local storage. The reduction in CPU load is due to the insertion operations are executed at both blockchain nodes and storage nodes, which distribute the computation loads. However, it is noted that the storage space reduction is at the cost of higher network usage and insertion latency caused by the propagation of the data objects in the DHT storage network. The memory usage and CPU load for *MapChain-D* are slightly higher than Ethereum with distributed storage due to the space and computation cost of the index chain. The experimental results support the theoretical complexity analysis in Section 3.4.

Table 3.3b shows that *MapChain-D* requires lower memory usage and CPU load during retrieval operations compared to Ethereum with local storage due to the distributed data storage and retrieval. Additionally, the *MapChain-D* model yields higher network usage and latency during data retrieval compared to Ethereum with local storage due to the lookup mechanism requiring additional network communications with the data chain network. This is considered as a tradeoff between communication complexity and resource consumption. The results show that the *MapChain-D* data retrieval outperforms Ethereum with local storage in terms of memory usage and CPU load at the cost of a higher network usage, which is consistent with the theoretical analysis in Section 3.4.

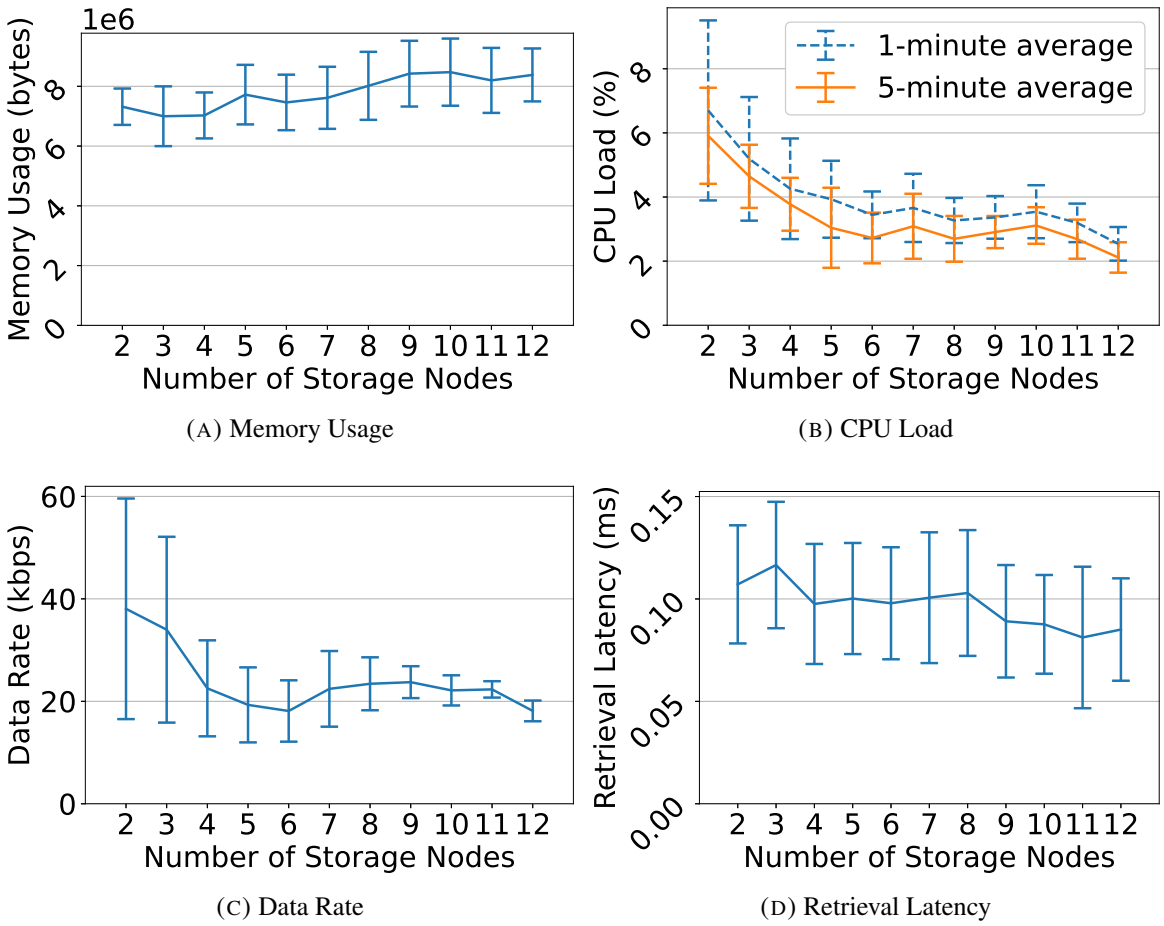
In Table 3.3b, it is also shown that compared to Ethereum with distributed storage, *MapChain-D* yields approximately 22% retrieval latency and 33% network usage at the cost of approximately 26% additional memory usage and 33% additional CPU load. The dual-chain structure successfully reduced the communication and time complexities of data retrieval by reducing the interactions with the distributed storage network compared to conventional single-chain with distributed storage.

3.6.2 Scalability Performance of *Mapchain-D*

Next, the scalability of *MapChain-D* is assessed with increasing DHT storage network size. To obtain the results, 500 data frames are inserted into the blockchain and the latency for each insertion operation is recorded. Then, the latest data frame from each of the four sensor nodes is retrieved starting from different block positions. The average retrieval latency across the four sensor nodes is calculated as the retrieval latency at the corresponding block position. 1-minute and 5-minute average CPU loads, network and memory usage for the storage nodes are also recorded in 10-second intervals to assess the computing and communication performance.

Number of storage nodes: Fig. 3.4a shows the impact of memory usage as increasing the number of DHT storage nodes. The figure shows that the average memory usage is relatively stable with an increasing number of storage nodes. This shows that the *MapChain-D* model efficiently distributes the data and memory usage across the storage nodes. Fig. 3.4b and Fig. 3.4c show that the average CPU load and data rate for each storage node decreases with an increasing number of storage nodes. This is because the blockchain node only needs to send the insertion query to one of the storage nodes. Therefore, the average processing power and bandwidth consumption for the insertion query decreases with an increasing number of storage nodes. Fig. 3.4d shows that the retrieval latency gradually decreases with an increasing number of storage nodes, which shows that the *MapChain-D* model provides good scalability in terms of retrieval latency. Fig. 3.4 highlights that *MapChain-D* provides excellent scalability with stable or decreasing performance as the number of DHT storage nodes increases.

k-bucket size: The size of k-bucket in Kademlia defines the number of other storage nodes known by each of the storage nodes in the network [14]. The redundancy level of data could be adjusted by using various k-bucket sizes where the "expected distance" between any pair of nodes is changed. The tests are repeated using various sizes of k-bucket to investigate the impact on the performance when the k-bucket size is changed. The k-bucket size is varied within the range between 2 and 12. The number of nodes is kept constant at 12. The rest of the parameters are also kept unchanged. In practice, the k-bucket size could be adjusted based



(A) Memory Usage

(B) CPU Load

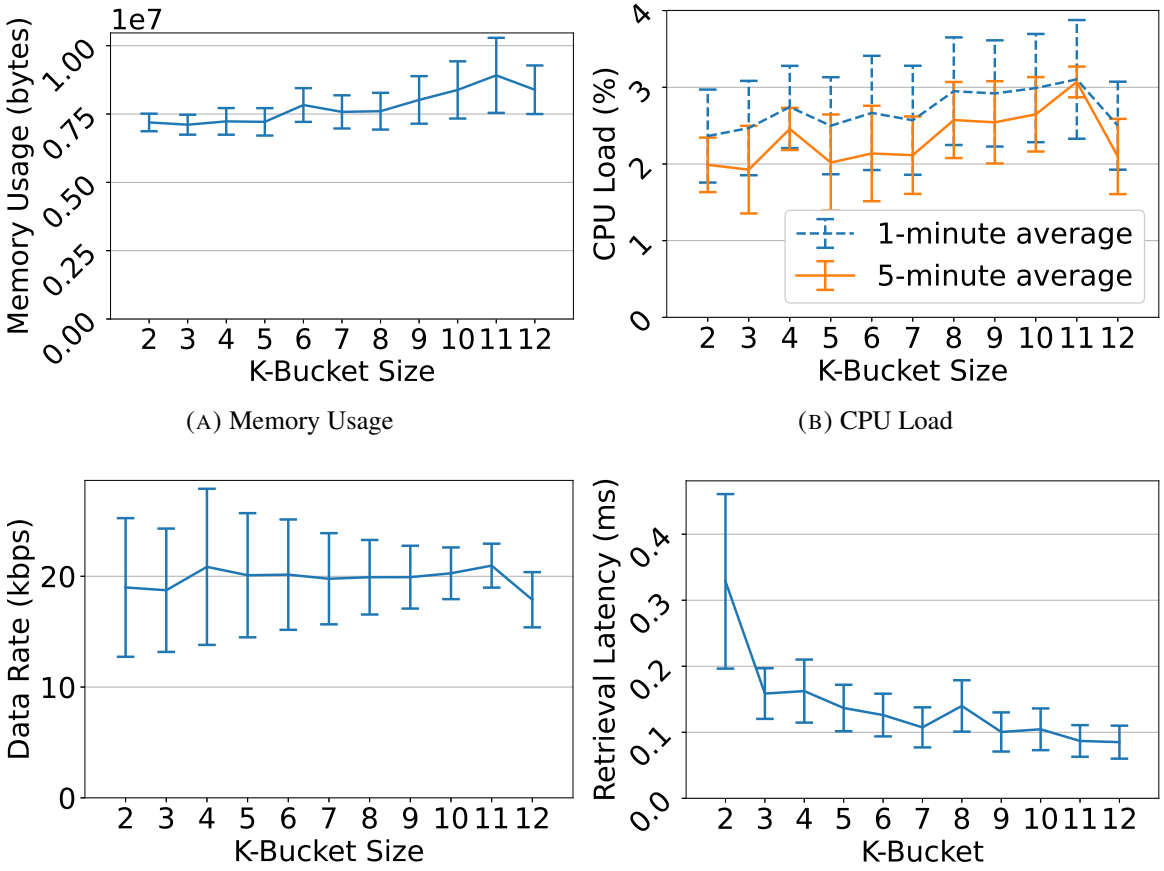
(C) Data Rate

(D) Retrieval Latency

FIGURE 3.4. System Performance for Different Numbers of Storage Nodes in *MapChain-D* Network

on multiple factors such as the level of network dynamic, reliability of the storage nodes, amount of available system resources and the requirement of success query rate.

Fig. 3.5a shows the impact of memory usage as increasing the k-bucket size in the DHT. The figure shows that the memory usage gradually increases with increasing k-bucket size. This is caused by the increased memory consumption for the routing table and redundancy level. Fig. 3.5b shows that the CPU load is relatively stable across different k-bucket sizes. Fig. 3.5c shows the variance in data rate decreases as the increasing k-bucket size. This is due to the fact that the redundancy level in the DHT increases with increasing k-bucket size. Fig 3.5d shows that the retrieval latency decreases as the increasing k-bucket size due to the increase in the redundancy level in the DHT with increasing k-bucket size.



(A) Memory Usage

(B) CPU Load

(C) Data Rate

(D) Retrieval Latency

FIGURE 3.5. System Performance for Different Numbers of Blockchain Nodes in *MapChain-D* Network

Number of blocks iterated during retrieval: Fig. 3.6 shows that the single-chain models generally yield a higher retrieval latency compared to the dual-chain models for various numbers of blocks iterated during the retrieval operation. It also shows that the single-chain DHT model results in the highest retrieval latency because the data lookup scheme needs to be conducted in the DHT storage network using the raw data. For dual-chain models, using either DHT (i.e., *MapChain-D*) or local storage can achieve a similar lowest retrieval latency due to their efficient data lookup schemes. Furthermore, *MapChain-D* requires less storage than dual-chain local storage models which highlights its advantage in resource-constrained IoT networks.

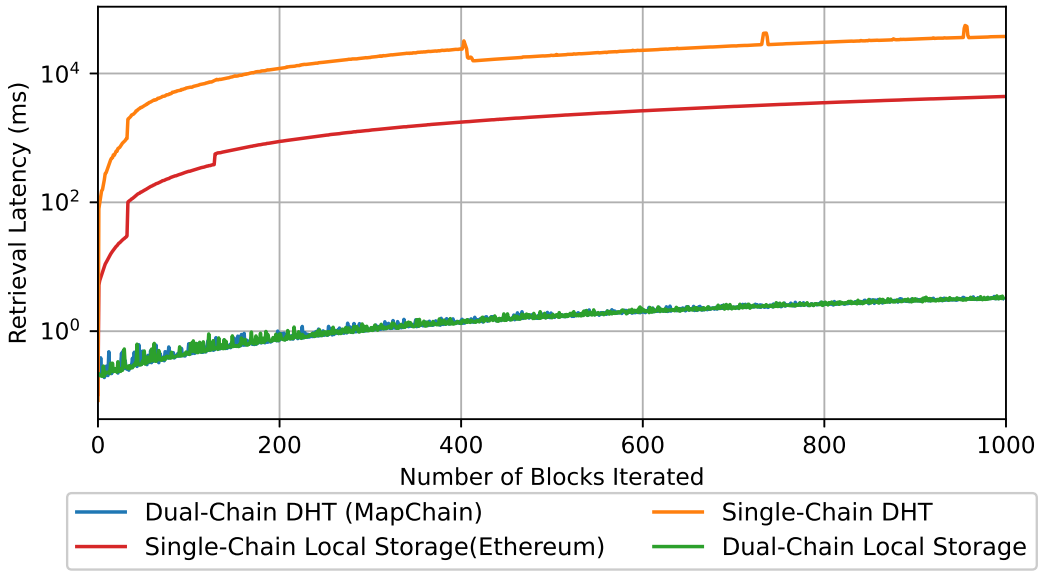


FIGURE 3.6. Retrieval Latency for Different Numbers of Blocks Iterated

3.7 Summary and Discussion

This chapter proposed and analysed a dual-blockchain distributed data management framework, named *MapChain-D*, for large-scale heterogeneous IoT systems. The system model and data exchange protocols were introduced and the performance of the distributed data insertion and retrieval protocols were analysed from the perspectives of space, time and communications complexities. To ensure that *MapChain-D* is suitable for practical IoT scenarios, a testbed implementation of *MapChain-D* was deployed. Experimental evaluations were performed with benchmark comparisons to conventional single-chain Ethereum solutions with local and distributed storage. Experimental results highlighted that the proposed *MapChain-D* framework outperformed single-chain Ethereum with local storage in terms of space and computation complexities with lower network usage and latency during retrieval compared to single-chain Ethereum with distributed storage. Therefore, *MapChain-D* is considered a solution that simultaneously reduces storage space consumption and computation usage and improves retrieval efficiency. Furthermore, the experimental results confirmed that *MapChain-D* provides a highly scalable design where the size of the data chain network can be increased

with relatively stable or decreasing impacts on the storage node memory consumption, CPU load, data rate, and retrieval latency.

CHAPTER 4

Hierarchical Federated Learning Model Verification

4.1 Introduction

In Chapter 3, a blockchain-based trusted distributed data storage system was proposed. After collecting and storing the sensor and actuator data, the next step is to explore more insightful patterns of the historical data. Utilising ML models such as ANNs has become an emerging approach to processing IoT data in practical applications. In order to improve efficiency and privacy compared to cloud-based training, DML approaches such as FL were proposed to offload model training from the centralised server to individual clients. In FL, each client performs model training locally using their own data before transferring their locally trained model parameters to the cloud server to generate an aggregated global model. All the participating clients contribute to the global model without needing to transfer their raw data to an external agent, and the global model benefits from all data available in the system. The aggregated FL model is generated using a weighted linear combination of client model parameters and transferred back to all clients after aggregation. In conventional FL, the cloud server determines the client weights applied to different client's model parameters, which might depend on the clients' dataset size, reputation and trust levels [64], [66].

In a conventional DML network, all the clients communicate with a single cloud server for model updates, creating scalability issues and single point-of-failure [48]. Hierarchical approaches such as HFL [48]–[51] were proposed to divide clients into multiple groups and introduce multiple edge servers as "local centres" to perform local aggregation using all client models within a group before sending the locally aggregated model to the cloud server for global aggregation. In HFL, additional trust and bias issues are introduced due to

the partial aggregation at multiple edge servers, which might have competing interests [59]. However, due to the isolation of the client devices in each group, none of the entities in the system has a full view of all local and global model aggregation processes. This makes it infeasible to utilise the conventional FL model verification scheme to guarantee the correctness of the global model aggregation results, which contains all the partial model aggregation results. Additionally, the conventional FL model verification approaches require complex key exchange or pseudo-random number generators. Extending it to the HFL architecture requires different key pairs or pseudo-random number generators for each group because the group model aggregation results need to be verified independently for each level of the hierarchy. The excessive computing and communication overheads for model integrity verification conflict with the aim of HFL to improve computing and communication efficiencies.

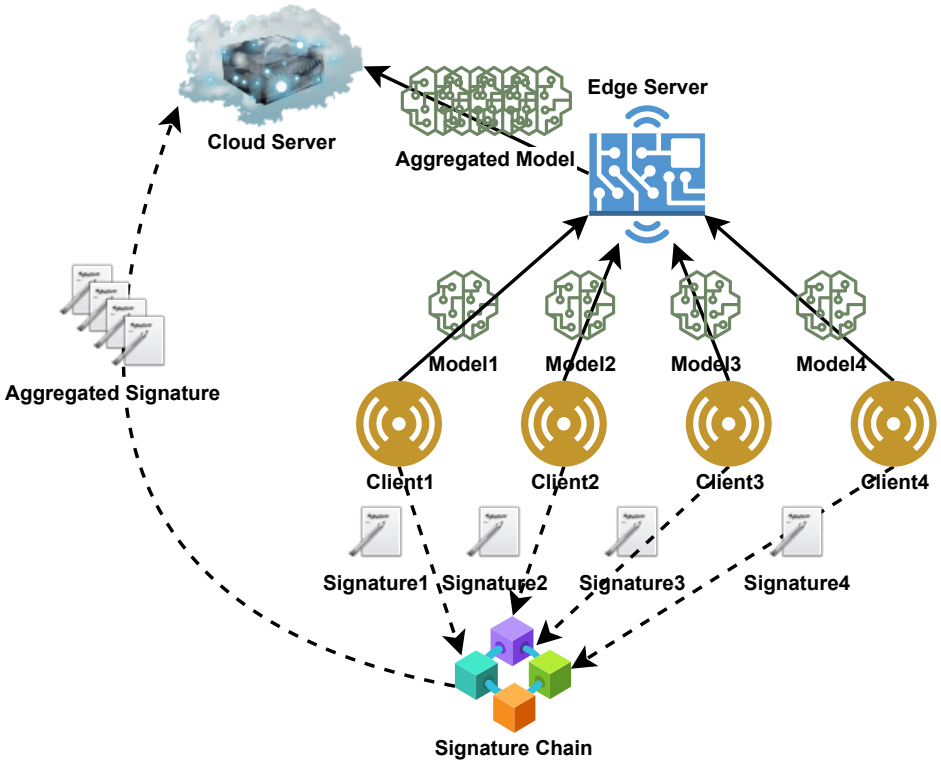


FIGURE 4.1. System Overview of the Proposed VHFL Framework

To address the scalability issue of the model verification in HFL, the *VHFL* technique is proposed in this chapter to leverage a blockchain to store model signatures, allowing any entity in the system to verify the globally or locally aggregated models at various levels of

the hierarchy. As illustrated in Fig. 4.1, the blockchain acts as the role of a trusted third party that stores the homomorphic-hash-based signatures of all client models. The tamper-proof feature of the blockchain enforces the integrity of the signatures, and the transparency ensures that each individual model signature is accessible to all the entities in the system. The aggregation servers or clients are able to query the blockchain for individual signatures or aggregated signatures to verify the integrity of clients' local models or the correctness of model aggregation results at different hierarchical levels.

4.2 Threat Model and Security Requirements

This Section presents the security threats we considered and the requirements for the proposed *VHFL* technique.

4.2.1 Weighted Aggregation in Hierarchical Federated Learning

For simplicity, a three-level HFL network is used as example in this chapter. The network includes clients, edge servers, and the cloud server. The hierarchical topology can be extended to multi-level networks with edge servers at different hierarchical layers. The most commonly used FedAvg[1] [10] is considered. Let N_g be the number of clients in a given edge group g, and $n_{g,m}$ be the size of the local dataset at a given client m in the group. In each FL training round, every client performs local training and sends their model parameters, denoted as $\mathbf{w}_{g,m}$, to the edge server for aggregation. Next, the edge server applies a client weight to each client model which is defined as:

$$e_{g,m} = \frac{n_{g,m}}{\sum_{m=1}^{N_g} n_{g,m}}. \tag{4.1}$$

[1]While FedAvg in Eq. (2.2) is considered as the federated aggregation algorithm in this chapter, the solution can be easily adjusted to other FL algorithms such as FedSGD by performing the weighted summation on the loss functions' results instead of the updated model weights. In this chapter, the model weights or model gradients are defined as *model parameters.*

Each edge server $\mathcal{E}_g$ sends its edge model parameters $\mathbf{w}_g$ to the cloud server, which is calculated as:

$$\mathbf{w}_g \leftarrow \sum_{m=1}^{N_g} \left(e_{g,m} \times \mathbf{w}_{g,m}\right). \tag{4.2}$$

After receiving the edge model parameters from G edge servers, the cloud server applies a client weight to each edge model parameter which is defined as:

$$e_g = \frac{\sum_{m=1}^{N_g} n_{g,m}}{\sum_{g=1}^{G}\sum_{m=1}^{N_g} n_{g,m}}. \tag{4.3}$$

The cloud server then multicasts the cloud model parameters $\mathbf{w}_c$ to all the clients which are calculated as:

$$\mathbf{w}_c \leftarrow \sum_{g=1}^{G} \left(e_g \times \mathbf{w}_g\right). \tag{4.4}$$

4.2.2 Threat Model

In the HFL framework, the model parameters or client weights can be maliciously modified before or after the aggregation process. These types of attacks are defined as a *model poisoning attack* or a *client weight attack*, respectively. Both attacks aim to significantly reduce model utility because the aggregation can be biased towards the wrong direction, or the convergence rate can be reduced. Whilst it is possible for devices at different hierarchy layers to collude and perform combined attacks, only one type of attack will be considered at a time to identify their impacts on the model utility clearly.

Model poisoning attack: The individual clients' model parameters or the aggregated model parameters can be changed by any aggregation server or during the communication to poison the aggregated models. It is assumed that the attackers are unable to access the clients' private datasets but know their model parameters. Therefore, the untargeted Gaussian attack adopted from [60] is considered, where the attacker randomly changes the model parameters sampled from a Gaussian distribution based on the range of the original model parameters before the attack.

Client weight attack: Attackers can also tamper with the client weights applied to the model aggregations [66], [67]. In such client weight attacks, the client weights are deliberately increased to bias the aggregated model parameters towards the attackers' interest and increase their dominance on the aggregated models. Client weight attacks are more difficult to detect as the client model parameters may remain unchanged, and the aggregation server might also be unaware of the change in client weights during the communication. To quantify the attack effectiveness, it is assumed that the aggregation servers hold a public dataset with similar properties to the clients' private datasets. The malicious aggregation server deliberately increases the weight of the model with the least validation accuracy using the public dataset to reduce the performance of the aggregated model.

4.2.3 Integrity and Confidentiality Requirements

The objective of *VHFL* is to efficiently protect the integrity of federated model aggregations while preserving client models' confidentiality. Therefore, the proposed *VHFL* technique needs to satisfy the following requirements:

- All model signatures (model hashes and client weights) must be immutable once generated. The model signatures and their generation and verification functions are transparent to all entities in the system.
- An aggregated model signature must be valid only if the model aggregation is performed using the correct model parameters and client weights.
- The model hashes in the signatures must be irreversible and collision-resistant such that it is computationally infeasible to generate the original model given its hash or generate an alternative model whilst keeping the corresponding hash unchanged.

Furthermore, the following specific assumptions are considered during the model integrity verification processes:

- Each client or server is unable to access the model parameters generated by clients or servers from different edge groups.

- All the clients are benign and generate valid signatures for their model parameters. All the benign aggregation servers generate the aggregated model parameters correctly.
- Each device has sufficient resources to perform model integrity verification, and can access all model signatures during the integrity verification process.

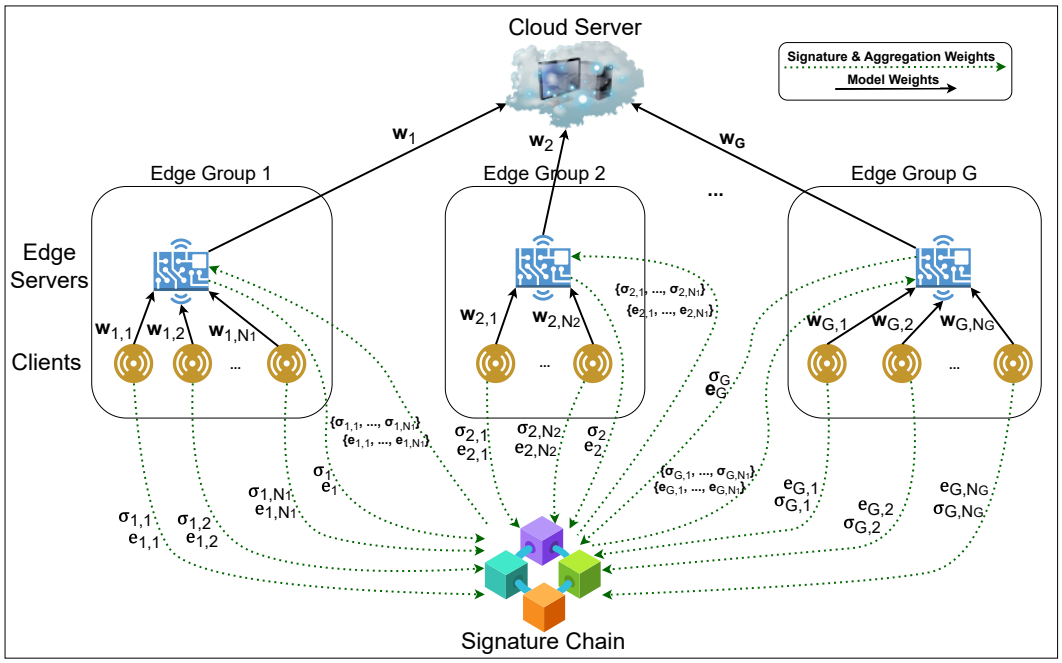
Based on the above requirements and assumptions, the algorithms for *VHFL* training, signature generation, and model integrity verification will be detailed in the following Section.

4.3 System Model

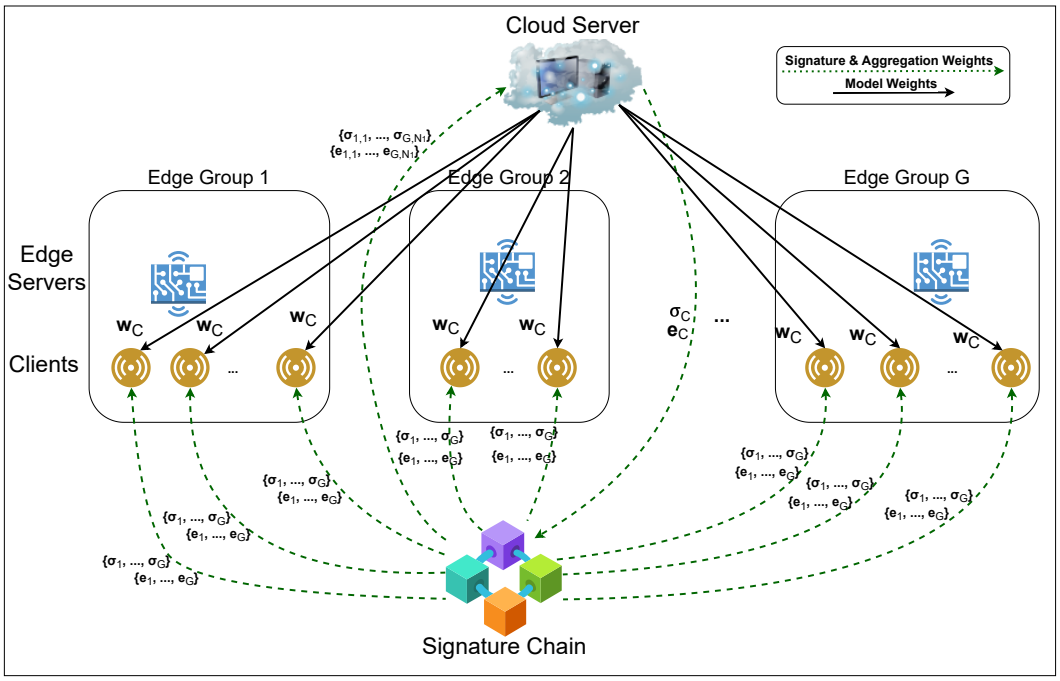
This Section introduces the *VHFL* technique where the system model illustrating the model exchange, aggregation and integrity verification processes at the edge and cloud server layers are shown in Fig. 4.2. A hybrid approach, that combines digital signatures and blockchain, is considered. First, the model parameters are digested to a hash value H using a common homomorphic hash function shared within the system. The hash is then concatenated with its corresponding client weight to form the model signature (i.e., Sig $= \big(H||e\big)$). Next, the model signature is stored in an immutable and transparent blockchain data structure (i.e., signature chain) to preserve its integrity. This allows all the entities to verify the integrity of any individual, edge, or cloud models. The clients provide their model parameters $\mathbf{w}_{g,m}$, where g ($g \in G$) is the edge group index and m ($m \in N_g$) is the client index, to their corresponding edge server $\mathcal{E}_g$. Each edge server $\mathcal{E}_g$ aggregates the model parameters from all the clients $\mathcal{D}_{g,m}\ \forall\ m \in N_g$ in edge group g and sends the edge model parameters $\mathbf{w}_g$ to the cloud server for cloud aggregation. Key notations are listed in Table 4.1.

4.3.1 Model Integrity Verification Process

The proposed *VHFL* model integrity verification processes for client, edge, and cloud models work as follows:



(A) edge aggregation



(B) cloud aggregation

FIGURE 4.2. VHFL Model Exchange, Aggregation, and Integrity Verification

Client model parameters integrity verification: For the m-th client in the g-th edge group, let $\mathbf{w}_{g,m}$ denote its trained model parameters and $\mathbf{w}'_{g,m}$ denote the model parameters received

TABLE 4.1. List of Notations

	Notation	Explanation
Constants	G	Number of edge groups
	$\{N_1, ..., N_G\}$	Number of clients in edge groups
Parameters for clients in edge group g	$\{\mathbf{w}_{g,1}, ..., \mathbf{w}_{g,N_g}\}$	Client model parameters
	$\{e_{g,1}, ..., e_{g,N_g}\}$	Edge client weights
	$\{H_{g,1}, ..., H_{g,N_g}\}$	Client model hashes
Parameters for edge groups	$\{\mathbf{w}_1, ..., \mathbf{w}_G\}$	Edge aggregated model parameters
	$\{e_1, ..., e_G\}$	Cloud client weights
	$\{H_1, ..., H_G\}$	Edge model hashes
Cloud parameters	$\mathbf{w}_c$	Cloud-aggregated model parameters
	H_c	Cloud model hash

by edge server $\mathcal{E}_g$ for edge aggregation. The model hash $H_{g,m}$ is generated by applying a non-reversible homomorphic hash function to the trained model parameters:

$$H_{g,m} = \texttt{Hash}\left(\mathbf{w}_{g,m}\right) \tag{4.5}$$

Assuming that $H_{g,m}$ cannot be tampered with after generation. $\mathbf{w}'_{g,m}$ are verified if $\texttt{Hash}\left(\mathbf{w}'_{g,m}\right) = H_{g,m}$.

Lemma 1: If any client model parameter in $\mathbf{w}_{g,m}$ is changed (i.e., $\mathbf{w}'_{g,m} \neq \mathbf{w}_{g,m}$), the hash generated by $\mathbf{w}'_{g,m}$ is different from $H_{g,m}$. Therefore, the model poisoning attacks targeting client model parameters can be detected.

Proof: According to the third requirement in Sec. 4.2.3, assume that it is computationally infeasible to generate an alternative model $\mathbf{w}'_{g,m} \neq \mathbf{w}_{g,m}$ to match a given model hash where $\texttt{Hash}\left(\mathbf{w}'_{g,m}\right) = H_{g,m}$. If any client model parameter in $\mathbf{w}_{g,m}$ is changed, the original model hash $H_{g,m}$ is not equal to the hash of the changed client model. Therefore, $\forall\ \mathbf{w}'_{g,m} \neq \mathbf{w}_{g,m}$, let $H'_{g,m} = \texttt{Hash}\left(\mathbf{w}'_{g,m}\right)$, then $H'_{g,m} \neq H_{g,m}$.

Edge Model Aggregation Integrity Verification: For the m-th client in g-th edge group, let $\mathbf{w}_{g,m}$ and $e_{g,m}$ denote its trained model parameters and edge client weight, respectively. Whereas $\mathbf{w}'_{g,m}$ and $e'_{g,m}$ are the model parameters and client weights used for the edge aggregation at edge server $\mathcal{E}_g$. Let $H_{g,m} = \texttt{Hash}\left(\mathbf{w}_{g,m}\right)$. Given N_g clients in the g-th edge

group, the edge aggregation integrity verification aims to verify whether the following edge model parameters $\mathbf{w}_g$:

$$\mathbf{w}_g = \sum_{m=1}^{N_g} \left(e'_{g,m} \times \mathbf{w}'_{g,m} \right) \tag{4.6}$$

is correct given all the client model hashes $H_{g,m}$ and client weights $e_{g,m}$ in g-th edge group. $\mathbf{w}_{g,m}$ can only be computed by the clients as they retain the training data. Each edge server $\mathcal{E}_g$ computes $\mathbf{w}_g$ using $\mathbf{w}'_{g,m}$ and $e'_{g,m}$ for all clients in the g-th edge group. Using the properties of homomorphic hashing for a linear combination given in [16], the following relationship can be defined for homomorphic hash values of the edge aggregated model parameters $\mathbf{w}_g$ and individual model parameters and client weights from N_g clients in the g-th edge group:

$$H_g = \texttt{Hash}\left(\mathbf{w}_g\right) = \texttt{Hash}\left(\sum_{m=1}^{N_g} \left(e'_{g,m} \times \mathbf{w}'_{g,m} \right) \right) = \prod_{m=1}^{N_g} \texttt{Hash}\left(\mathbf{w}'_{g,m} \right)^{e'_{g,m}} \tag{4.7}$$

If $\mathbf{w}'_{g,m} = \mathbf{w}_{g,m}$ and $e'_{g,m} = e_{g,m}\ \forall\ m \in [1, N_g]$, the edge aggregation is performed using the correct model parameters and client weights. Substitute them into Eq. (4.7):

$$H_g = \prod_{m=1}^{N_g} \texttt{Hash}\left(\mathbf{w}'_{g,m} \right)^{e'_{g,m}} = \prod_{m=1}^{N_g} \texttt{Hash}\left(\mathbf{w}_{g,m} \right)^{e_{g,m}} \tag{4.8}$$

Substitute Eq. (4.5) into Eq. (4.8), the edge aggregated model parameters in $\mathbf{w}_g$ are valid if the following holds:

$$H_g = \texttt{Hash}\left(\mathbf{w}_g\right) = \prod_{m=1}^{N_g} \left(H_{g,m} \right)^{e_{g,m}} \tag{4.9}$$

Using Eq. (4.9), one would need only the edge model parameters $\mathbf{w}_g$, and client model signatures $\{\text{Sig}_{g,1}, ..., \text{Sig}_{g,N_g}\}$ (where $\text{Sig}_{g,m} = \left(H_{g,m} || e_{g,m}\right)$) to verify the integrity of an edge model. Individual client model parameters $\{\mathbf{w}_{g,1}, ..., \mathbf{w}_{g,N_g}\}$ will remain private.

Theorem 1: If any client model parameter in $\mathbf{w}_{g,m}$ or edge client weight $e_{g,m}$ in edge group g is changed during the edge aggregation, the edge aggregated hash H_g is also changed. Therefore, edge aggregation attacks can be detected.

Proof: If Eq. (4.8) holds, $H_g = \texttt{Hash}\left(\sum_{m=1}^{N_g}\left(e_{g,m} \times \mathbf{w}_{g,m}\right)\right)$. Assume that $\forall\ e'_{g,m} \neq e_{g,m}$ or $\mathbf{w}'_{g,m} \neq \mathbf{w}_{g,m}$, $\sum_{m=1}^{N_g}\left(e'_{g,m} \times \mathbf{w}'_{g,m}\right) \neq \sum_{m=1}^{N_g}\left(e_{g,m} \times \mathbf{w}_{g,m}\right)$. According to the third requirement in Sec. 4.2.3, assume that it is computationally infeasible to vary the model parameters whilst keeping the hash unchanged. Therefore, let $H'_g = \texttt{Hash}\left(\sum_{m=1}^{N_g}\left(e'_{g,m} \times \mathbf{w}'_{g,m}\right)\right)$, $\forall\ \sum_{m=1}^{N_g}\left(e'_{g,m} \times \mathbf{w}'_{g,m}\right) \neq \sum_{m=1}^{N_g}\left(e_{g,m} \times \mathbf{w}_{g,m}\right)$, $H'_g \neq H_g$. This means that if any client model parameter $\mathbf{w}_{g,m}$ or edge client weight $e_{g,m}$ is changed, the hash value of the edge model will not be equal to its edge aggregated hash H_g calculated using Eq. (4.9).

Cloud model aggregation integrity verification: For the g-th edge group, let $\mathbf{w}_g$ and e_g denote its edge aggregated model parameters and cloud client weight, respectively. Whereas $\mathbf{w}'_g$ and e'_g are the model parameters and client weights used for the cloud aggregation at the cloud server. Given G edge groups, the cloud aggregation integrity verification aims to verify whether the following cloud-aggregated model parameters $\mathbf{w}_c$:

$$\mathbf{w}_c = \sum_{g=1}^{G}\left(e'_g \times \mathbf{w}'_g\right) \tag{4.10}$$

is correct given all edge model hashes H_g and client weights e_g. $\mathbf{w}_g$ can only be computed by the edge servers as they receive and aggregate client models within the corresponding edge group. The cloud server computes $\mathbf{w}_c$ using $\mathbf{w}'_g$ and e'_g for all edge servers. Similar to edge integrity verification, the following relationship can be derived using the properties of homomorphic hashing:

$$H_c = \texttt{Hash}\left(\mathbf{w}_c\right) = \texttt{Hash}\left(\sum_{g=1}^{G}\left(e'_g \times \mathbf{w}'_g\right)\right) = \prod_{g=1}^{G}\texttt{Hash}\left(\mathbf{w}'_g\right)^{e'_g} \tag{4.11}$$

If $\mathbf{w}'_g = \mathbf{w}_g$ and $e'_g = e_g\ \forall\ g \in [1, G]$, the cloud aggregation is performed using the correct edge aggregated model parameters and client weights. Substitute them into Eq. (4.11):

$$H_c = \prod_{g=1}^{G}\texttt{Hash}\left(\mathbf{w}'_g\right)^{e'_g} = \prod_{g=1}^{G}\texttt{Hash}\left(\mathbf{w}_g\right)^{e_g} \tag{4.12}$$

Substitute Eq. (4.9) into Eq. (4.12), the cloud-aggregated model parameters $\mathbf{w}_c$ is valid if the following holds:

$$H_c = \texttt{Hash}\left(\mathbf{w}_c\right) = \prod_{g=1}^{G} \left(H_g\right)^{e_g} \tag{4.13}$$

To verify the integrity of the cloud model, one would need only the cloud model parameters $\mathbf{w}_c$, cloud client weights $\{e_1, ..., e_G\}$, and edge aggregated model hashes $\{H_1, ..., H_G\}$. There is no need to retrieve any of the edge model parameters $\{\mathbf{w}_1, ..., \mathbf{w}_G\}$. Note that the edge aggregated model hashes can be calculated using the client model signatures by Eq. (4.9), if the edge aggregation is not performed correctly (i.e., Eq. (4.9) does not hold $\forall\ g \in [1, G]$), the cloud model integrity verification will be failed as a result.

Theorem 2: If any edge model parameter $\mathbf{w}_g$ or cloud client weight e_g is changed, the cloud-aggregated hash H_c is also changed. Therefore, cloud aggregation attacks can be detected.

Proof: If Eq. (4.12) holds, $H_c = \texttt{Hash}\left(\sum_{g=1}^{G}\left(e_g \times \mathbf{w}_g\right)\right)$. Assume that $\forall\ e'_g \neq e_g$ or $\mathbf{w}'_g \neq \mathbf{w}_g$, $\sum_{g=1}^{G}\left(e'_g \times \mathbf{w}'_g\right) \neq \sum_{g=1}^{G}\left(e_g \times \mathbf{w}_g\right)$. As per the third requirement in Sec. 4.2.3, it is computationally infeasible to vary the model parameters whilst keeping the hash unchanged. Therefore, let $H'_c = \texttt{Hash}\left(\sum_{g=1}^{G}\left(e'_g \times \mathbf{w}'_g\right)\right)$, $\forall\ \sum_{g=1}^{G}\left(e'_g \times \mathbf{w}'_g\right) \neq \sum_{g=1}^{G}\left(e_g \times \mathbf{w}_g\right)$, $H'_c \neq H_c$. This means that if any edge aggregated model parameter $\mathbf{w}_g$ or cloud client weight e_g is changed, the hash value of the cloud model will not be equal to the original cloud-aggregated hash H_c calculated using Eq. (4.13).

4.3.2 Algorithms for VHFL Technique

Algorithms 3-5 define the proposed *VHFL* hierarchical model training, aggregation, and integrity verification technique at different hierarchical layers. The algorithms can be executed repeatedly until the termination criteria (i.e., performance, training time, training rounds, etc.) are met.

Algorithm 3 is executed at all the clients $\mathcal{D}_{g,m}$ in parallel after the clients first join the system or receive the model update from the cloud server. Line 8 checks for cloud-aggregated model

Algorithm 3 VHFL Client-Side Model Training, Signature Generation, and Model Integrity Verification

Initialisation:

1: • Learning rate: η
• Client datasets in g-th edge group: $\{\mathbf{d}_{g,1}, ..., \mathbf{d}_{g,N_g}\}$

2: **for** $g \; in \; 1, ..., G$ **do in parallel**
3: **for** $m \; in \; 1, ..., N_g$ **do in parallel** ▷ At client (g, m)
4: Retrieve $\text{Sig}_{g,m} \; \forall \; g \in [1, G], \; m \in [1, N_g]$ from signature chain and extract $H_{g,m}$ and $\mathbf{e}_{g,m}$ from $\text{Sig}_{g,m}$
5: **if** $\mathbf{w}_{g,m} == \varnothing$ or $\mathbf{w}_c == \varnothing$ **then**
6: Initialise $\mathbf{w}_{g,m}$
7: $\mathbf{w}_{g,m} \leftarrow \mathbf{w}_{g,m} - \eta \nabla \texttt{Loss} \left(\mathbf{w}_{g,m}; \mathbf{d}_{g,m} \right)$
8: **else if** $\texttt{Hash} \left(\mathbf{w}_c \right) == \prod_{g=1}^{G} \left(\prod_{m=1}^{N_g} \left(H_{g,m} \right)^{e_{g,m}} \right)^{e_g}$ **or** $\mathbf{w}_c$ has higher accuracy than $\mathbf{w}_{g,m}$ **then**
9: $\mathbf{w}_{g,m} \leftarrow \mathbf{w}_c$
10: $\mathbf{w}_{g,m} \leftarrow \mathbf{w}_{g,m} - \eta \nabla \texttt{Loss} \left(\mathbf{w}_{g,m}; \mathbf{d}_{g,m} \right)$
11: $H_{g,m} \leftarrow \texttt{Hash} \left(\mathbf{w}_{g,m} \right)$
12: **else**
13: Revert to $\mathbf{w}_c$ at the previous training round
14: **end if**
15: Send $\mathbf{w}_{g,m}$ to edge server $\mathcal{E}_g$
16: Send $\text{Sig}_{g,m} = \left(H_{g,m} || e_{g,m} \right)$ to the signature chain
17: **end for**
18: **end for**

parameters or client weights changes at the previous training round. A client accepts the model update if the cloud signature is valid. Additionally, if the validation accuracy of the cloud model is higher than that of a client's local model, the model update is still accepted to ensure that the model training would not be prevented by an insignificant change in the cloud model parameters that does not negatively affect the model utility.

If the client-side model integrity verification fails, the client model parameters will be reverted to the ones from the previous training round. Lines 10-11 are for model training and hash generation. Finally, each client will upload its model signature (concatenated model hash and client weights) to the signature chain and send their model parameters to their corresponding edge servers.

Algorithm 4 VHFL Edge Server-Side Signature Verification and Model Aggregation

1: **for** $g \; in \; 1, ..., G$ **do in parallel** ▷ At edge server (g)
2: Receive $\{\mathbf{w}_{g,1}, ..., \mathbf{w}_{g,N_g}\}$ from clients in group g
3: **for all** $m \; in \; 1, ..., N_g$ **do**
4: Retrieve $\text{Sig}_{g,m}$ from the signature chain
5: Extract $H_{g,m}$ from $\text{Sig}_{g,m}$
6: **if** $\texttt{Hash}\left(\mathbf{w}_{g,m}\right) != H_{g,m}$ **then**
7: Revert to $\mathbf{w}_{g,m}$ at the previous training round
8: **end if**
9: **end for**
10: $\mathbf{w}_g \leftarrow \sum_{m=1}^{N_g} \left(e'_{g,m} \times \mathbf{w}'_{g,m}\right)$
11: Send $\mathbf{w}_g$ to the cloud server
12: **end for**

Algorithm 4 is executed at all edge servers $\mathcal{E}_g$ in parallel after all the clients in the corresponding edge group have finalised model training and signature generation (i.e., Algorithm 3 has been completed). If a client model fails to validate the hash (i.e., line 6 is true, indicating that the client model parameters have been tampered with), the corresponding client model will be reverted to the one at the previous training round. It then accumulates the updated or reverted client model parameters. Compared to the cloud signature verification in Algorithm 3, edge servers cannot validate the model accuracy as they cannot access clients' datasets.

Algorithm 5 VHFL Cloud Server-Side Signature Verification and Model Aggregation

1: Receive $\{\mathbf{w}_1, ..., \mathbf{w}_G\}$ from edge servers
2: **for all** $g \; in \; 1, ..., G$ **do**
3: Retrieve $\text{Sig}_{g,m} \; \forall \; m \; in \; 1, ..., N_g$ from the signature chain
4: Extract $H_{g,m}$ and $\mathbf{e}_{g,m}$ from $\text{Sig}_{g,m}$, $\forall \; m \; \in \; [1, N_g]$
5: **if** $\texttt{Hash}\left(\mathbf{w}_g\right) != \prod_{m=1}^{N_g} \left(H_{g,m}\right)^{e_{g,m}}$ **then**
6: Revert to $\mathbf{w}_g$ at the previous training round
7: **end if**
8: **end for**
9: $\mathbf{w}_c \leftarrow \sum_{g=1}^{G} \left(e'_g \times \mathbf{w}'_g\right)$
10: Multicast $\mathbf{w}_g$ to all clients

Algorithm 5 is executed at the cloud server after all edge servers finish their edge model aggregation (i.e., Algorithm 4 has been completed). If an edge model fails signature validation (i.e., line 5 is true, indicating that any of the edge aggregated model parameters or edge client weights have been tampered with), the corresponding edge model will be reverted to the

parameters from the previous training round. It then accumulates the updated or reverted edge aggregated model parameters. Like Algorithm 4, the cloud server cannot validate model accuracy as it cannot access clients' datasets.

4.3.3 Acquiring Model Signatures for Integrity Verification

One must obtain the corresponding model signatures to verify the integrity of a model. Due to the assumption that the integrity verification function is known to everyone in the *VHFL* system, ensuring that the model signatures are immutable after being generated is essential. If we can avoid the signatures and algorithms being tampered with, the attack success rate will be drastically reduced. It is assumed that it is computationally infeasible to obtain any client model parameters given the corresponding model signature. Therefore, any permissionless blockchain can be leveraged as the signature chain in *VHFL* system model to maximise transparency and enforce the integrity of the model signatures. A smart contract can be utilised to enforce the integrity of the signature generation and integrity verification algorithm. The signature aggregation processes can be executed within the blockchain network using the smart contract to reduce the computation load at the servers and clients. In contrast to existing solutions that store all the client model parameters in a blockchain, storing model signatures significantly reduces the storage size and computation cost of blockchain transactions because the model signatures are significantly smaller in size than the entire matrix of model parameters.

4.3.4 Scalability Analysis

As illustrated in Fig. 4.3, the proposed *VHFL* processes include blockchain interactions (in dashed red arrows), signature generation and integrity verification processes (in dashed green arrows), and conventional HFL model training and aggregation processes (in solid orange arrows). The figure shows that the hash generation or verification and model training or aggregation are performed on a separate process executed simultaneously on each client, edge server, and the cloud server. This means that the proposed *VHFL* has parallel processing

not only at different devices, but also at each device. There is no need for a client, an edge server, or the cloud server to wait for the integrity verification results to perform training or aggregation. Instead, if the signature verification fails, they will stop their model training or aggregation and revert their model parameters to the latest previous model with a valid signature.

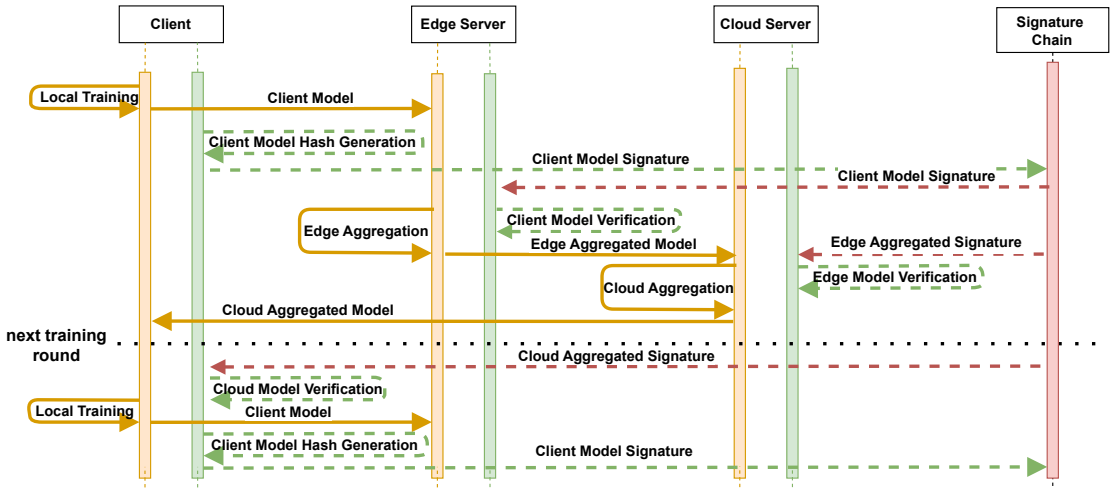


FIGURE 4.3. VHFL Sequence Diagram Illustrating the Proposed *VHFL* Processes

Client training, signature generation, and integrity verification: It is assumed that all the clients start their training simultaneously. Because Algorithm 3 executes at all clients $\mathcal{D}_{g,m}$ in parallel, the time complexity is independent of the number of clients in the system. Instead, it depends on the time taken by the last client that finishes cloud model integrity verification, local training, and hash generation processes. Note that the time complexity for each client depends on the longer of the two processes for local training and hash generation (Algorithm 3, lines 10-11) and cloud signature verification (Algorithm 3, line 8).

Edge integrity verification and aggregation: Because Algorithm 4 is executed at all edge servers $\mathcal{E}_g$ in parallel, the time complexity is independent of the number of edge servers in the system. If all the edge servers start at the same time and all the edge servers have the same computing and communication capability, the time complexity for edge aggregation depends

only on the client model size and the maximum number of clients in each edge group:

$$O(|\mathbf{w}|\times \max\left(N_1, ..., N_G\right)) \tag{4.14}$$

In a practical scenario, the total time complexity for edge aggregation depends on the last edge server that finishes edge aggregation and signature verification. Note that the time complexity for each edge server would be the longer of the two processes for edge aggregation (Algorithm 4, line 10) and client model hash verification (Algorithm 4, line 6).

Cloud integrity verification and aggregation: Given the computing and communication capability of the cloud server, the time complexity for cloud aggregation only depends on the size of edge models (which is the same as client models) and the number of edge groups:

$$O\left(|\mathbf{w}|\times G\right) \tag{4.15}$$

It would be the longer of the two processes for cloud aggregation (Algorithm 5, line 9) and edge signature verification (Algorithm 5, line 5).

4.4 Experimental Setup

As the cloud server, edge servers, and clients accept the edge and cloud models only when Eq. (4.9) and Eq. (4.13) hold, the correctness of our *VHFL* technique is established. To evaluate the proposed *VHFL* technique, we simulate attacks applied to model parameters and client weights at various hierarchical layers and evaluate the accuracy of the integrity verification algorithm as well as the utility of the aggregated models. The simulation setup is shown in Fig. 4.4, where the training server simulates all the training and aggregation operations as the clients, edge servers and the cloud server. The model signatures are transmitted to one of the blockchain nodes after the execution of each of the operations. For performance overhead evaluation, we increase the number of clients and edge groups and record the computation time and resource usage. For simplicity, a three-layer hierarchical structure similar to Fig. 4.2 is simulated. It is assumed that all the clients and edge servers send model updates on time.

4.4.1 Dataset

To simulate non-independent and identically distributed (non-IID) characteristics of client data, FEMNIST [180] – a preprocessed version of the Extended MNIST (EMNIST) [181] dataset is considered. Only the samples of digits in the FEMNIST dataset are used. Therefore, the task of the FL model is to classify the hand-written letters into ten classes of digits. A subset of the FEMNIST dataset containing samples from one writer is assigned to each client sequentially for local training and another writer for model validation. The writers allocated to each client remain the same for all training rounds.

The simulations are repeated using IID data samples for some selected attack scenarios. A commonly used CIFAR-10 dataset [182] is considered and randomly partitioned into equally-sized subsets for all the clients to simulate IID data. For each client, samples of the images from different categories are allocated. The task is to classify the images into ten categories.

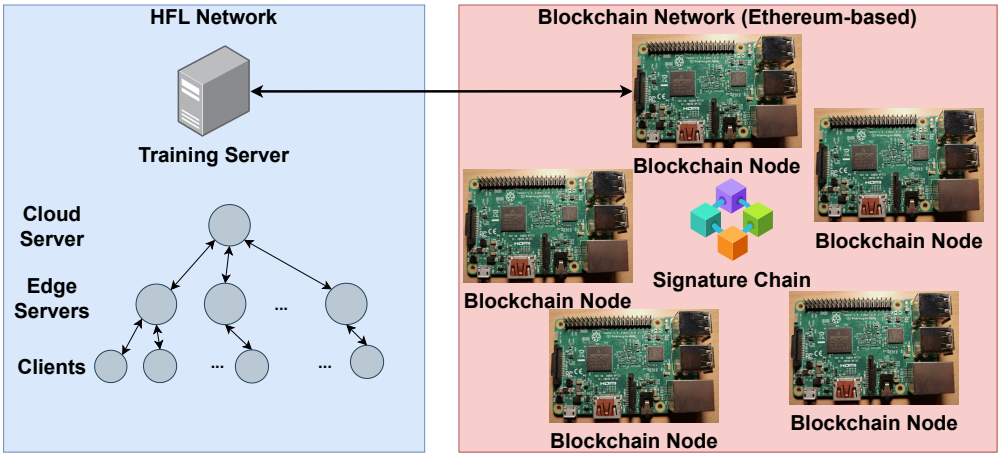


FIGURE 4.4. Overview of the VHFL Simulation Setup

4.4.2 Neural Network Configurations

To assess the performance and resource overheads of the proposed *VHFL* technique, the test neural network model should yield a relatively good model utility and be suitable to be trained in resource-constraint environments. After some trial runs, a simple CNN model is selected for the FEMNIST dataset. The CNN model contains the following five layers: 2D convolution

(32 output filters, kernel size 3, ReLU activation), 2D max pooling (pool size 2, with padding), dropout layer (dropout rate 0.25), dense layer (32 outputs, ReLU activation), and dense layer (10 outputs, Sigmoid activation). For CIFAR-10, a more complex and commonly used LeNet CNN structure is considered.

The Adam optimiser is used for both CNN models with the learning rate set to 0.0005. The cross-entropy loss function is used, and the categorical accuracy (i.e., frequency of matched prediction) applied to the prediction of the validation set is used as the evaluation metric for model utility. It is assumed that all the clients are honest during the model evaluation process. Each client performs model evaluation using their validation set, and the average categorical accuracy is recorded as the validation accuracy. The CNN model training simulations are conducted on the TensorFlow platform[2] written in Python. The integrity verification and attack simulations are performed on a workstation with an Intel Core i9-12900 CPU and 128 GiB of memory. The batch size is set to 50.

4.4.3 Blockchain Network

The experimental setup for the blockchain network simulation separates the clients from other entities to assess the resource overheads of *VHFL* in a practical environment. In addition to the clients and aggregation servers, a private Ethereum network is deployed using five RPi nodes with Go Ethereum (Geth)[3] library installed. Clique PoA[4] consensus algorithm is utilised to avoid excessive power consumption. The blockchain node the clients connect to is also the PoA signer responsible for signing and creating new blocks. The block time is set to five seconds such that one block is expected to be generated every five seconds.

The Web3.py[5] library is utilised to connect to an Ethereum node via a web socket. A smart contract is developed using Solidity language for the blockchain-based persistent storage and retrieval of all the historical model signatures. For benchmark comparison, the smart contract

[2]https://www.tensorflow.org/
[3]https://geth.ethereum.org/
[4]https://eips.ethereum.org/EIPS/eip-225
[5]https://web3py.readthedocs.io/en/stable/

stores the model parameters instead. The smart contract is shared between clients and servers, and it executes once it receives a model update from a client or server. The blockchain latency and gas use[6] to execute the smart contract on the blockchain are recorded as the blockchain overheads.

4.4.4 Integrity Verification Sensitivity

The model parameters verification results can be classified into the following four sets:

- $\mathbf{W}_{\text{TP}}$: weight $\mathbf{w}$ is correct and signature Sig is valid
- $\mathbf{W}_{\text{TN}}$: weight $\mathbf{w}$ is incorrect and signature Sig is invalid
- $\mathbf{W}_{\text{FP}}$: weight $\mathbf{w}$ is incorrect but signature Sig is valid
- $\mathbf{W}_{\text{FN}}$: weight $\mathbf{w}$ is correct but signature Sig is invalid

Based on the above sets, the Integrity Verification Sensitivity (IVS) for *VHFL* is defined as:

$$\text{IVS} = \frac{|\mathbf{W}_{\text{TP}}|+|\mathbf{W}_{\text{TN}}|}{|\mathbf{W}_{\text{TP}}|+|\mathbf{W}_{\text{TN}}|+|\mathbf{W}_{\text{FP}}|+|\mathbf{W}_{\text{FN}}|}, \tag{4.16}$$

which is a measure of the effectiveness of *VHFL* in detecting modification attacks on the model or client weights at the client, edge or cloud layers.

4.4.5 Overall Computing Time Overheads

Since the experiment is performed on a single training server sequentially, the overall computation time for a single training round needs to be calculated using the time taken for model training, aggregation, hash generation and integrity verification executed on each client, edge server or cloud server recorded during the simulation.

Table 4.2 lists the notations used to calculate the time overhead of *VHFL* for hash generation and verification processes at each client, edge server and the cloud server. It is assumed that the clients only start a new training round after receiving an updated model from the

[6]Gas is the accounting mechanism in Ethereum that determines the computational complexity of a transaction.

TABLE 4.2. Time Notations for Overhead Calculations

Device	Notation	Explanation
Client $\mathcal{D}_{g,m}$	$t_{\text{train}}(g,m)$	Local training
	$t_{\text{ct\_gen}}(g,m)$	Client model hash generation
	$t_{\text{cd\_ver}}(g,m)$	Cloud model integrity verification
Edge Server $\mathcal{E}_g$	$t_{\text{e\_aggr}}(g)$	Edge aggregation
	$t_{\text{ct\_ver}}(g)$	Client model integrity verification
Cloud Server	$t_{\text{c\_aggr}}$	Cloud aggregation
	$t_{\text{e\_ver}}$	Edge model integrity verification

cloud server and there is no dropped-out client or server in the system. The edge and cloud signature aggregation processes are performed by the blockchain nodes, but they are included in the integrity verification at clients and the cloud server for simplicity. The calculations of computing overheads are detailed as follows.

Client-side overheads: In conventional HFL, the clients only perform local training such that:

$$t_{\text{ct,HFL}}\left(g,m\right) = t_{\text{train}}\left(g,m\right). \tag{4.17}$$

For group g, it is assumed that all the clients start their local training simultaneously. The overall client-side time consumption would be the maximum time consumption for a client in group g to perform local training, which is:

$$t_{\text{ct,HFL}}\left(g\right) = \max\left(\{t_{\text{ct,HFL}}(g,m)\ \forall\ m\ \in\ [1,N_g]\}\right). \tag{4.18}$$

In *VHFL*, all the clients need to verify the integrity of the cloud-aggregated model after the initial training round. Nonetheless, the local training can be performed in parallel with the cloud model integrity verification process, such that the local training time with *VHFL* is $\max\left(t_{\text{train}}\left(g,m\right), t_{\text{cd\_ver}}\left(g,m\right)\right)$. The client model hash generation is performed after the local training and cloud model integrity verification processes at a time consumption of $t_{\text{ct\_gen}}\left(g,m\right)$. As such, the overall client-side time consumption for a training round with *VHFL* at client (g,m) is calculated as:

$$t_{\text{ct,VHFL}}\left(g,m\right) = \max\left(t_{\text{train}}\left(g,m\right), t_{\text{cd\_ver}}\left(g,m\right)\right) + t_{\text{ct\_gen}}\left(g,m\right). \tag{4.19}$$

Since the local training and cloud model integrity verification processes for all the clients $\mathcal{D}_{g,m}$ in group g are performed in parallel, the total client-side time consumption is the maximum time consumption across the local training and cloud model integrity verification processes performed at all the clients $\mathcal{D}_{g,m}$ in group g. Based on Eq. (4.19), the local training time with *VHFL* can be evaluated as:

$$t_{\text{ct,VHFL}}(g) = \max\left(\{t_{\text{ct,VHFL}}\left(g, m\right)\ \forall\, m\ \in\ [1, N_g]\}\right). \tag{4.20}$$

Overheads at each edge group: In conventional HFL, an edge server starts an edge aggregation process after receiving the local training results from all the clients in its corresponding edge group, which is at $t_{\text{ct,HFL}}(g)$. Based on Eq. (4.18), the overall time consumption at edge group (g) is:

$$t_{\text{e,HFL}}(g) = t_{\text{ct,HFL}}(g) + t_{\text{e\_aggr}}(g), \tag{4.21}$$

where $t_{\text{e\_aggr}}$ is the edge aggregation time.

For *VHFL*, the edge server starts the edge aggregation and client model integrity verification processes after all the client model signatures are generated. Based on Eq. (4.20), the overall time consumption at edge group (g) is:

$$t_{\text{e,VHFL}}(g) = t_{\text{ct,VHFL}}(g) + \max\left(t_{\text{e\_aggr}}\left(g\right), t_{\text{ct\_ver}}\left(g\right)\right), \tag{4.22}$$

where the edge aggregation is performed in parallel with the client model integrity verification processes such that the edge aggregation time with *VHFL* is calculated as $\max(t_{\text{e\_aggr}}(g), t_{\text{ct\_ver}}(g))$.

Overall overheads: In conventional HFL, the cloud server starts the cloud aggregation process after receiving the edge models from all the edge groups. As such, the overall time consumption for HFL is given by:

$$t_{\text{c,HFL}} = \max\left(\{t_{\text{e,HFL}}(g)\ \forall\, g\ \in\ [1, G]\}\right) + t_{\text{c\_aggr}}, \tag{4.23}$$

where $t_{\text{e,HFL}}(g)$ is defined in Eq. (4.21) and $t_{\text{c\_aggr}}$ is the cloud aggregation time.

For *VHFL*, the cloud server starts the cloud aggregation and edge model integrity verification processes after all the edge model signatures are generated. The cloud aggregation is performed in parallel with the edge model integrity verification processes. Therefore, the overall time consumption with *VHFL* is calculated as:

$$t_{\text{c,VHFL}} = \max\left(\{t_{\text{e,VHFL}}(g)\ \forall\ g\ \in\ [1, G]\}\right) + \max(t_{\text{c\_aggr}}, t_{\text{e\_ver}}), \tag{4.24}$$

where $t_{\text{e,VHFL}}(g)$ is defined in Eq. (4.22).

Finally, the overall computing time overhead for *VHFL* relative to the conventional HFL is calculated as:

$$\text{overhead\_c} = \frac{t_{\text{c,VHFL}} - t_{\text{c,HFL}}}{t_{\text{c,HFL}}}. \tag{4.25}$$

4.5 Experimental Results

This Section presents and discusses the results from the simulations based on the setup in Sec. 4.4.

4.5.1 Integrity Verification Sensitivity and Overheads

In the experiment, 750 clients are simulated and the model is trained using the FEMNIST dataset, which is the maximum number of clients the dataset can be split under the experimental configurations. The average IVSs are calculated using Eq. (4.16) and the computing overheads are evaluated based on Section 4.4.5. Both values are recorded for different levels of hash precision, which is defined as the number of decimal places used for generating the model hash signatures [7].

Integrity verification sensitivity: For the client model poisoning attack, up to 1000 random model parameters are changed from each client's model. After each change, the edge servers

[7] To evaluate the IVS, the change in model or client weights is limited to one decimal point (i.e., random numbers sampled between zero and one)

TABLE 4.3. Simulation Evaluation of Integrity Verification Sensitivity (IVS) and Computing Time Overheads for *VHFL* with Different Hash Precision Levels

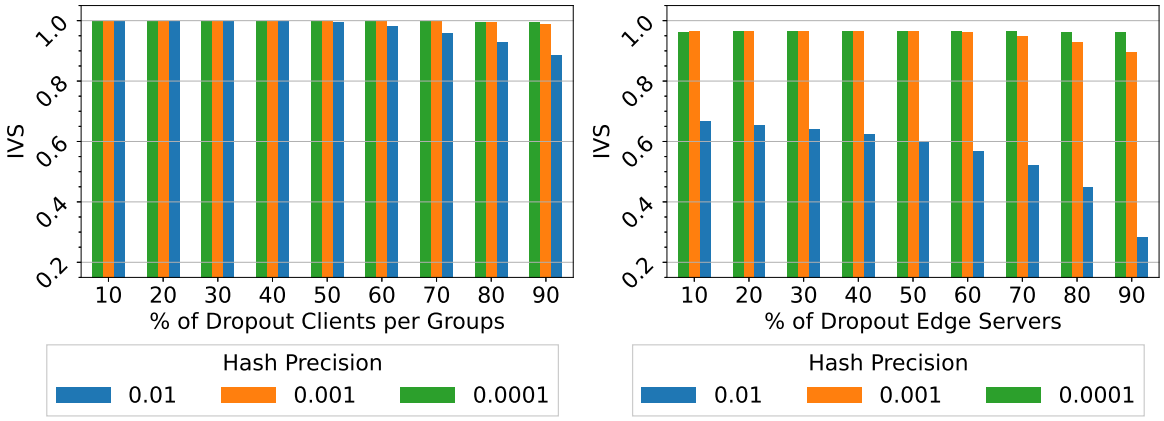
Signature Hash Precision	Average IVS Client	Edge	Cloud	Overall Computing Time Overhead
10^{-1}	0.709	0.405	0.036	1.74
10^{-2}	0.950	0.985	0.716	1.86
10^{-3}	0.991	0.999	0.999	1.98

will verify the integrity of all the client models. As shown in Table 4.3, increasing the hash precision from 10^{-1} to 10^{-2} results in a significant increase in the IVS from 0.709 to 0.950.

For the edge client weight attack, the 750 clients are divided into 25 edge groups, where each edge group contains 30 clients. Then, the edge client weights are changed for different numbers of randomly selected clients in each edge group. After that, the integrity of edge models is verified. Table 4.3 shows that, similar to the client IVS, the IVS for edge model integrity verification increases with an increase in hash precision and at least two decimal places for the signature hash is required to achieve a high IVS of 0.985.

For the cloud client weight attack, the 750 clients are divided into 30 edge groups, each containing 25 clients. This ensures that the number of edge groups is equal to the group size in the edge client weight attack simulation enabling a fair comparison. Then, the cloud client weights are changed for different numbers of randomly selected edge groups. Table 4.3 shows that the signature hash precision for the cloud model should be at least three decimal places to achieve an IVS of 0.999 which indicates that the cloud aggregation attacks are more difficult to detect compared to client and edge attacks.

Dropout rates: In order to evaluate the robustness of *VHFL* in practical scenarios, the IVSs are recorded under different dropout rates of the clients or edge servers for edge integrity verification and cloud integrity verification accordingly. A dropout of a client or an edge server means that the client or edge server is unable to provide model parameter updates to the edge or cloud server for model aggregation. The model signatures of the dropout clients or edge groups would not be included during the signature aggregation, and there is no need to recompute any signature. In Fig. 4.5a, we see that the edge aggregation integrity



(A) Edge aggregation verification.

(B) Cloud aggregation verification.

FIGURE 4.5. VHFL Model Aggregation Integrity Verification Sensitivity (IVS) for Different Dropout Rates

verification is robust under client dropout even at a low hash precision of 10^{-2}, whereas the cloud aggregation integrity verification in Fig. 4.5b is less robust to the edge server dropout at hash precision 10^{-2}. The reason is that only a single cloud model gets generated for each training round as opposed to multiple edge models. Hence, the total number of client weights that are changed during the cloud aggregation is less than the number of changes during the edge aggregation. Clearly, a higher signature precision is required for the cloud model integrity verification to obtain similar sensitivity as the edge model integrity verification. If the hash precision is increased to 10^{-3} or 10^{-4}, both edge and cloud aggregation integrity verifications provide high sensitivity (i.e., greater than 0.89) even when 90% of the clients or edge servers are disconnected. The simulations show that *VHFL* can accurately detect aggregation attacks even under severely unstable network conditions.

Overall computing time overheads: The time consumption values for *VHFL* and HFL are evaluated for all the clients and edge servers in the simulation setup. Simulation evaluations for the overall computing time overhead of *VHFL* for different signature hash precision levels are listed in Table 4.3. The simulation results in Table 4.3 show that while higher hash precision increases the IVS of *VHFL*, this security advantage comes at the cost of higher computing time overheads for hash generation and verification. Interestingly, it is noted that increasing the hash precision from 10^{-1} to 10^{-3} can achieve a significant improvement in

the IVS of up to 28 times with only a small increase of 13.8% in the overall computing time overheads.

4.5.2 Attack Simulation for Model Utility

The attack simulations are performed repeatedly using the *VHFL* technique and HFL without model integrity verification while the rest of the configurations remain unchanged. The model aggregations are performed after each training round, and the attackers aim to reduce the model utility (i.e., validation accuracy) by changing the model parameters or client weights at different hierarchical layers. The clients then perform an evaluation using the cloud model after 100 training rounds. To ensure that the experiment is reproducible, the random seed is manually set. The simulation is repeated using ten different random seeds and the mean and percentiles are recorded as presented in the next set of figures. For simplicity, ten edge groups are considered where each edge group contains ten clients. First, the number of edge groups under attack is varied within the range between zero and nine. Then, the number of clients' weights under attack is changed to one, five, and ten. The simulations are repeated with different numbers of edge groups for attacks applied to client model parameters or edge client weights. Note that the attacks applied to the cloud client weights are unaffected by the number of clients under attack in each group as they are performed at the cloud layer. The experiment was repeated using the non-IID FEMNIST and IID CIFAR-10 datasets. It is expected that the validation accuracy for the CIFAR-10 dataset is lower than the FEMNIST dataset because the convergence rate for the CIFAR-10 dataset is lower than that for the FEMNIST dataset during the initial trial runs.

Model poisoning attacks: Model poisoning attacks are considered to be applied to the parameters of the clients' individual models according to Section 4.2.2. The attacks are performed before model aggregation (e.g., during the communications), where it is assumed that everyone intends to transmit their correct model parameters and signatures. Fig. 4.6 shows that HFL without integrity verification yields lower validation accuracy than *VHFL* for the FEMNIST dataset. For attacks with a lower portion of client models under attack, the non-IID characteristic of the dataset partially preserves the model utility for HFL. With the increase in

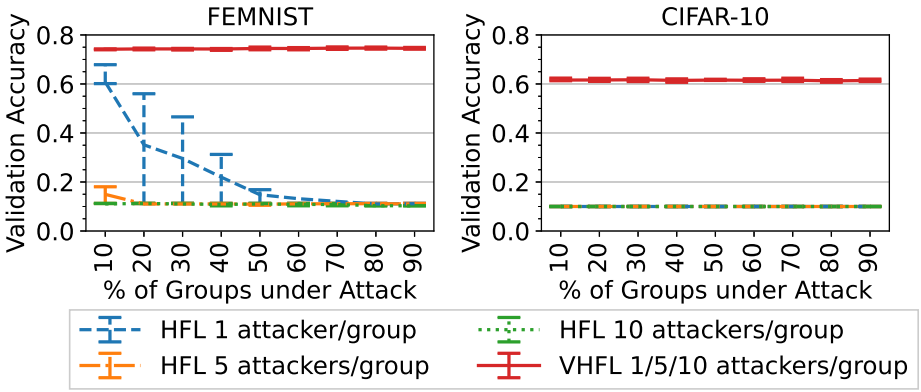


FIGURE 4.6. Model Utility Under Client Model Poisoning Attack with Different Numbers of Attackers per Group

the percentage of groups under attack and the number of clients in a group under attack, the validation accuracy for HFL decreases towards a random guess (i.e., 10% validation accuracy). Whereas *VHFL* always provides average validation accuracy for different numbers of groups under attack.

For the CIFAR-10 dataset, Fig. 4.6 shows that the validation accuracy for HFL is close to a random guess for all scenarios of client models under attack. Alternatively, *VHFL* constantly provides higher validation accuracy. Note that for both non-IID and IID datasets, *VHFL* recovers the validation accuracy and yields closer validation accuracy than HFL without an attack. It is concluded that *VHFL* successfully preserves the model utility under the client model poisoning attack. We also note that the client model poisoning attack has a more substantial impact on the IID dataset than the non-IID dataset because the non-IID dataset provides more statistical heterogeneity where the model parameters contributed by the non-malicious clients are still usable.

TABLE 4.4. Comparison with Multi-Krum Robust Aggregation Under Model Poisoning Attack

Attack Defencing Technique	% Validation Accuracy	
	FEMNIST	CIFAR-10
No Defence	14.76	10
Multi-Krum [97]	52.76	48.84
VHFL	74.43	61.52

The proposed *VHFL* is also compared with a state-of-the-art Multi-Krum robust aggregation algorithm [97]. The simulations are repeated using Multi-Krum aggregation with the number of Byzantine workers f set to be two, four and six. The rest of the configurations are identical to the HFL attack simulations without any attack defence technique. The average validation accuracies for different numbers of Byzantine workers and different numbers of clients and groups under attack using FEMNIST and CIFAR-10 datasets are shown in Table 4.4. It is observed that applying Multi-Krum improves the utility of the models in terms of validation accuracy and applying *VHFL* further increases the validation accuracy, compared to the scenario without any attack defence.

Client weight attacks: For client weight attacks at edge servers or the cloud server, the client weights of selected models are multiplied by random numbers greater than one. The rest of the client weights remain unchanged. We consider a scenario where lower-performing models trained with a randomly-labelled dataset are given higher client weights. Fig. 4.7 shows that for both non-IID and IID datasets, with the increasing percentage of groups under attack and the number of clients in a group under attack, the validation accuracy decreases at a higher rate for HFL without validation. Alternatively, *VHFL* provides more stable validation accuracy. However, the difference between the validation accuracy for HFL and *VHFL* is less for the IID dataset. The reason is that the increase in client weight of the worst-performed model reduces the portion of the model parameters trained by the unique characteristics of the non-IID dataset, whereas the variance of the model parameters trained by different clients is less for the IID dataset. We see that *VHFL* recovers the validation accuracy to a higher value compared to HFL without any attack defence. Therefore, *VHFL* is considered successfully preserves the model utility under edge client weight attacks.

For cloud client weight attack, Fig. 4.8 shows that for the non-IID dataset, the HFL model always yields a lower validation accuracy than *VHFL*. For the IID dataset, the average validation accuracy when 10% and 20% groups are under attack is lower for *VHFL*, and the range of the percentiles is also lower. For both scenarios, with the increasing percentage of groups under attack, the validation accuracy of HFL decreases towards a lower value.

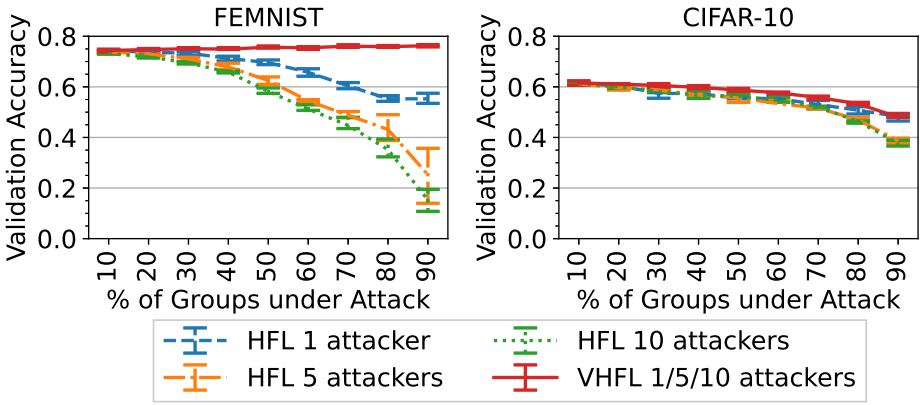


FIGURE 4.7. Model Utility Under Edge Client Weight Attack with Different Numbers of Attackers per Group

Therefore, *VHFL* can better preserve model utility under cloud client weight attacks. However, it is not as strong as in the case of edge client weight attacks.

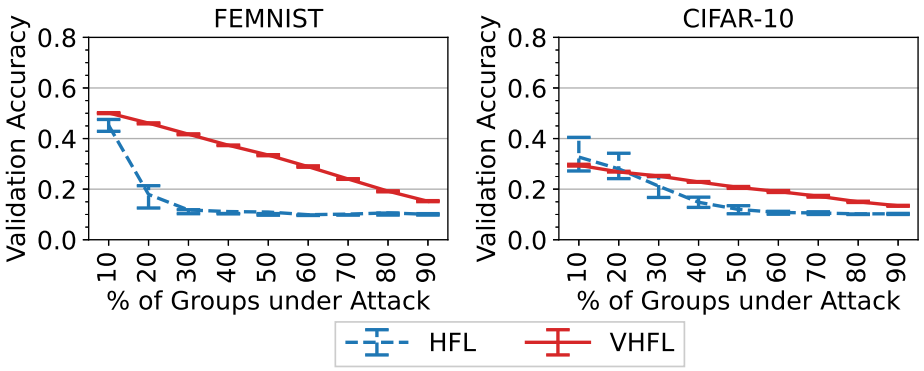
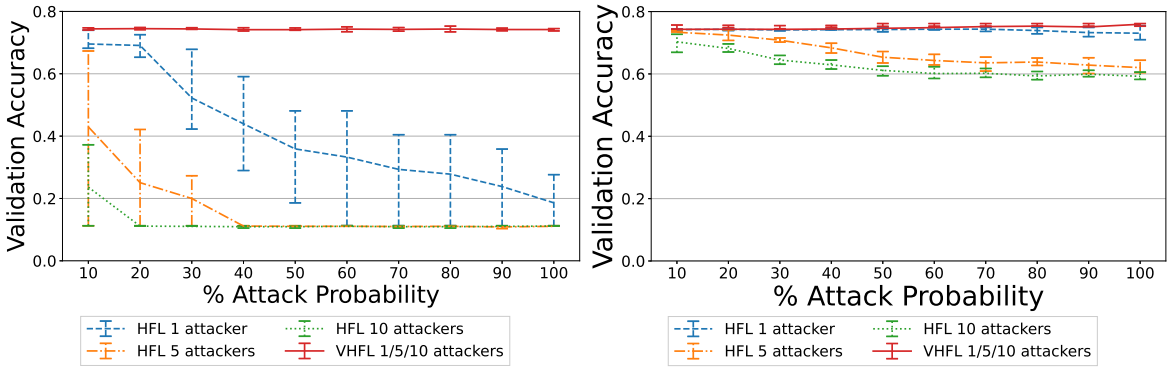


FIGURE 4.8. Model Utility Under Cloud Client Weight Attack with Different Datasets

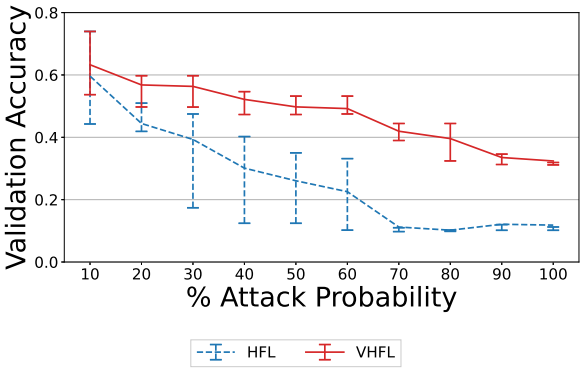
Attack probability: As the non-IID FEMNIST dataset is more suitable for the FL scenario with statistical heterogeneity, it is chosen to further analyse the model utility under occasional attacks with different probabilities (i.e., the number of times a client or server is expected to perform an attack). For attacks applied to the client models (i.e., Figs. 4.9a and 4.9b), different numbers of attacking clients in each edge group are considered. For attacks applied to the edge models (i.e., Fig. 4.9c), half of the edge groups are considered under attack. Figs. 4.9a and 4.9c show that for client model poisoning attacks and cloud client weight attacks. When the attack probability reaches 100%, HFL's performance is close to a random guess, whereas

VHFL always yields higher and more stable validation accuracy than HFL. For HFL, with the increasing number of clients or edge groups under attack, the validation accuracy decreases at a higher rate. For the edge client weight attack, Fig. 4.9b shows that the reduction in the validation accuracy is less than that for the other two attacks. The reason is that the change in a single cloud client weight has a similar effect as the change in the edge client weight of an entire edge group. Therefore, the edge client weight attack has a less significant impact on the model utility than the cloud client weight attack.



(A) Client model poisoning attack.

(B) Edge client weight attack.



(C) Cloud client weight attack.

FIGURE 4.9. Model Utility Under Attacks at Different Attack Probabilities (FEMNIST)

4.5.3 Blockchain Simulations

To assess the blockchain resource overheads of *VHFL*, 750 clients are divided into 30 edge groups of 25 clients. No malicious behaviour is simulated for blockchain simulation because

the blockchain overhead for a malicious client is the same as for an honest client. The experiment is repeated three times, and the average values are considered.

Blockchain latency: In Table 4.5a, the latency for blockchain model signature storage used in the *VHFL* technique is compared with conventional blockchain model parameter storage. The blockchain latency is divided into insertion and retrieval latencies for each client, edge server, and cloud server. It is assumed that there is no bandwidth constraint such that the blockchain retrieval operation can be performed in parallel. It is observed that the latency for blockchain model signature storage is lower than that for blockchain model parameter storage. This is because the signatures stored in the blockchain are much smaller in size compared to the model parameters.

Gas use: As shown in Table 4.5b, *VHFL* has the lowest gas use values for all the operations. This means that *VHFL* signature verification is more cost-effective to be deployed on the Ethereum blockchains and the associated computing overhead is less. Like the latency, the less gas use values are due to the much smaller storage size of the model signature compared to the model parameter.

TABLE 4.5. Simulation Evaluation of VHFL Blockchain Overheads for Insertion and Retrieval Operations Compared with Conventional Non-Signature Blockchain-Based Model Aggregation

(A) Comparison of blockchain latency time (s)

	Client		Edge		Cloud	
	Insertion	Retrieval	Insertion	Retrieval	Insertion	Retrieval
Conventional	0.51	0.043	0.57	0.046	6.00	0.047
VHFL	0.24	0.002	0.33	0.034	4.59	0.036
% Improvement	52.2	95.4	41.3	25.2	23.4	21.9

(B) Comparison of blockchain gas use ($\times 10^4$)

	Client		Edge		Cloud	
	Insertion	Retrieval	Insertion	Retrieval	Insertion	Retrieval
Conventional	17.64	3.75	17.55	94.13	17.83	2823.93
VHFL	6.68	0.11	6.69	66.48	4.39	1994.52
% Improvement	62.1	97.2	61.9	29.4	75.4	29.4

4.6 Summary and Discussion

This chapter proposed and analysed an integrity verification technique for HFL architectures named *VHFL*. *VHFL* employs lightweight, homomorphic hash-based FL model signatures that can be aggregated at different hierarchical layers to verify the integrity of the model aggregation at both the cloud and edge layers. First, the integrity verification sensitivity was evaluated for different hash precision levels and dropout rates to investigate the trade-off between the integrity verification sensitivity and computing time. Attack simulation results showed that the *VHFL* technique could detect malicious aggregation results and recover the model utility to achieve higher validation accuracy compared to HFL without integrity verification and the conventional Multi-Krum robust aggregation scheme. Specifically, *VHFL* could effectively increase the global HFL model utility under the model poisoning attack and client weight attack performed at different hierarchical layers. Furthermore, a practical testbed for blockchain simulations was deployed to assess the blockchain overhead. Results showed that *VHFL* effectively reduces both the blockchain latency and transaction fees compared with the benchmark blockchain-based model storage scheme.

CHAPTER 5

Flexible Layer Selection for Distributed Machine Learning

5.1 Introduction

In Chapter 4, a verification technique for DML model aggregation was proposed and analysed. The proposed *VHFL* technique effectively protects the integrity of HFL model aggregation and ensures that the edge and cloud aggregation servers perform model aggregation correctly. While model aggregation integrity is protected, privacy leakage during the sharing of model parameters becomes another concern due to the fact that some information about the clients' private training data is embedded in the model parameters. For example, in DRA, an adversary can reconstruct clients' private data used for model training by using the gradients during back propagation or comparing the model weights between two consecutive model updates [56], [101], [103]–[105].

SL [11], [12] was proposed to train a partially shared model at multiple clients in a semi-decentralised manner. In a basic SL, the ANN model is split into two parts at a split layer where the clients train the layers closer to the input layer, and the cloud server trains the rest of the layers. The activation outputs from the split layer are transferred from clients to the server during forward propagation. Then, the server performs server-side training and returns split-layer gradients to the clients so they can propagate their client-side models. In addition to offloading part of the computation resource consumption for model training from clients to the cloud server, SL also aims to address such privacy issues caused by transferring all the model weights between clients and the server.

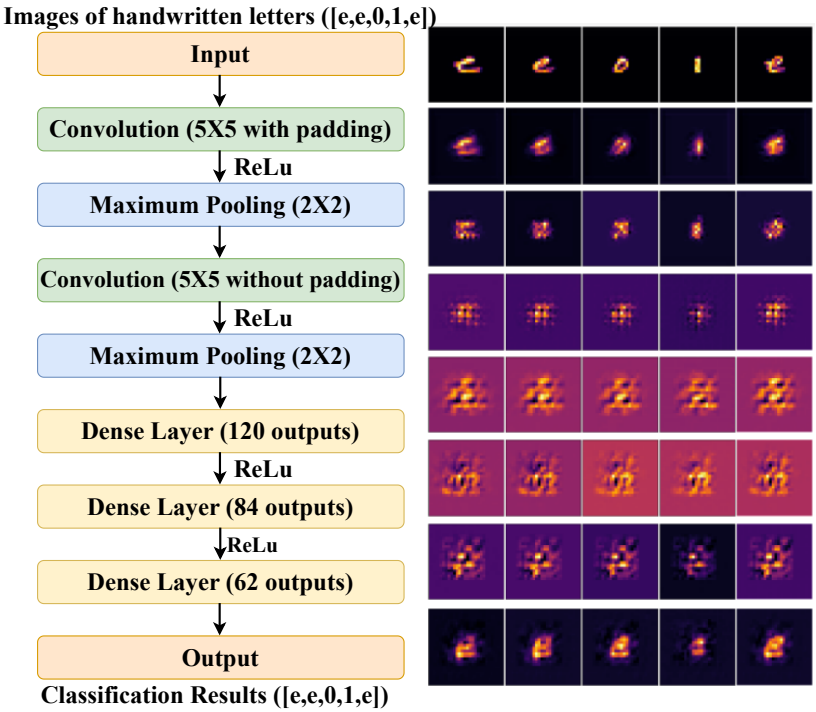


FIGURE 5.1. Information Extracted by Applying the Inference-Attack Model in [110] to the Outputs of the Neurons in Different Split Layers of a LeNet CNN

There have been a number of studies on privacy attacks in SL aiming to reveal the input features [109], [110], [140], [144], [145], [183]–[186], output labels [144], [146] or properties of the input features [110] using the split layer activation outputs or gradients transferred between clients and the server. Fig. 5.1 shows an example of how the input feature inference attack model in [110] can be used to extract different amounts of information from the outputs of the neurons in different split layers of the LeNet CNN. The figure highlights that layers closer to input data are more vulnerable to privacy attacks. The authors in [17], [187], [188] also show that the vulnerability of the privacy attack differs for different layers in an ANN model. However, in conventional SL, the server determines the model structure and split layer, and the individual clients cannot control their split layer according to their privacy requirements.

In a large-scale network, different groups of clients may have various privacy levels and information they wish to protect. Fixing the split layer in conventional SL hinders the ability

of clients to decide their levels of private information shared with other entities in the network by varying the amount of information shared with the server. To address this issue, a federated-split distributed learning framework named *FlexSplit* is proposed in this chapter, allowing each client to select their preferred split layer to control their individual privacy levels. Each client in the proposed *FlexSplit* framework selects a trusted edge server to perform part of the model training and determines their split layer according to their resource constraints and privacy requirements. Compared to the conventional single-server split learning setup, *FlexSplit* also improves scalability by performing partial model training processes at multiple edge servers in parallel before performing layer-wise aggregation at the cloud server.

5.2 System Model

Fig. 5.2 illustrates the proposed hierarchical *FlexSplit*'s system architecture. Dashed arrows indicate the transfer of split layer outputs and gradients during training, whereas solid arrows indicate the transfer of server-side model weights during aggregation. A client trains its client-side model and shares the split-layer output with its edge server, which in turn updates the server-side model and shares it with the cloud server for aggregation. Then, edge servers and the cloud server send updated models downwards, enabling a backward pass of the model gradients at the lower layers of the hierarchy (see Sec. 5.2.1 for details). The proposed *FlexSplit* framework allows a client to control the number of layers shared in the system by selecting an edge server with a suitable split layer that meets its privacy and utility requirements.

In *FlexSplit* design, it is assumed that a trusted edge server is always available for a client to perform training at the required split layer, and edge servers have sufficient computation and communication resources to perform server-side training. If an edge server's communication and computation resources reach their limit, it should not accept more clients. It is also assumed that the cloud server could be honest-but-curious, and could collude with one or more clients or edge servers to perform privacy attacks.

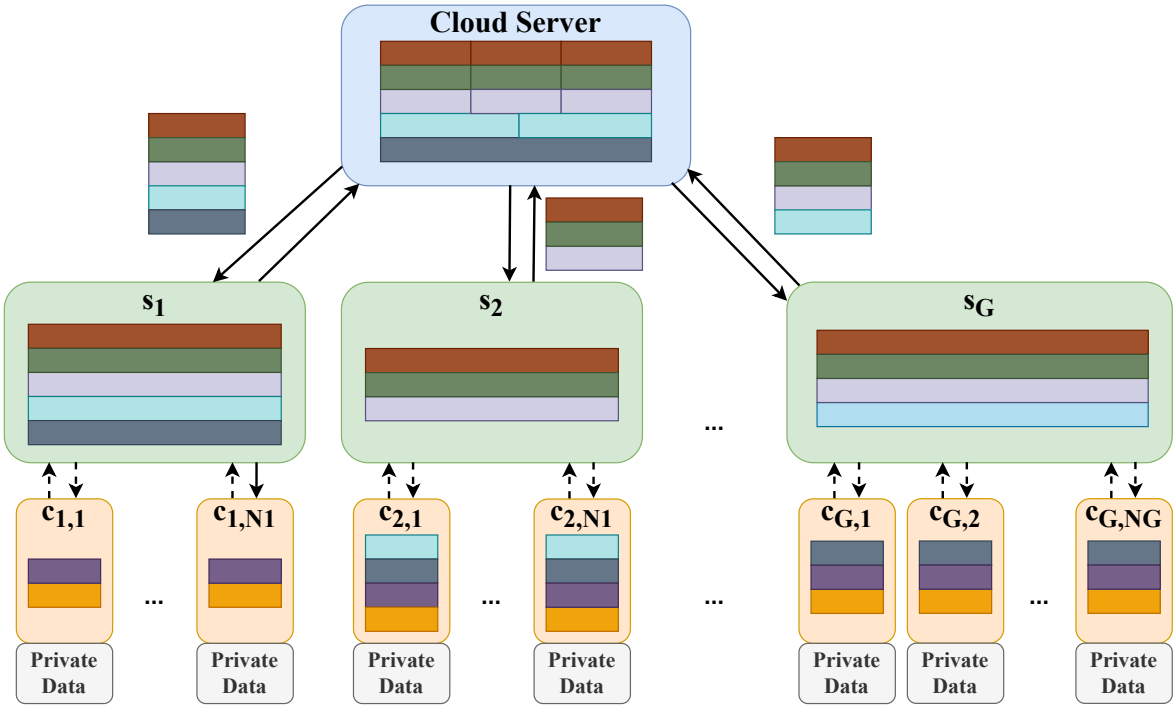


FIGURE 5.2. *FlexSplit* System Model (s – edge servers, c – clients)

5.2.1 Split-Aggregate Algorithm

Algorithm 6 summarises the split-aggregate-based model training in *FlexSplit*. In Algorithm 6, `ForwardPass` is the model's forward propagation function that computes the outputs or loss functions. `BackwardPass` is the model's backward propagation function that updates the model parameters using the losses or outputs from the latter layer. `SendtoClient`, `SendtoEdge` and `SendtoCloud` functions send data to clients, edge servers, and the cloud server, respectively. `ReceivefromClient`, `ReceivefromEdge` and `ReceivefromCloud` functions receive data from any client, edge server, or the cloud server, respectively. g and m denote the index of edge group and client, respectively.

Part of the algorithm is executed at each client (lines 4-9) and edge server (line 10-22) in parallel. First, a client performs a forward pass on its client-side model (line 5) and transfers the split layer outputs and labels of the training data to the edge server (line 6). When edge server $\mathcal{E}_g$ receives data from client $\mathcal{D}_{g,m}$ in the edge group (line 11), it performs a forward and backward pass of its server-side model using the data received from the client (lines

12-13), followed by transferring the updated split layer back to the client (line 14) and an update of the aggregation counter (line 15). The client that receives an updated split layer performs a backward pass of the client-side model (lines 7-8). Meanwhile, if the number of client training rounds reaches the threshold $t\_threshold_g$ (line 16), the edge server transfers the updated server-side model to the cloud server (line 18) for aggregation (lines 23-31). To improve the model utility, layer-wise model aggregation is introduced (lines 25-27) at the cloud server to aggregate the server-side models with different numbers of layers. Thus, the cloud server aggregates the updated server-side model using the weighted average according to the total size of the datasets used for the current training round (n in line 24).

Note that the aggregation process is asynchronous, as the cloud server does not need to wait till all the edge servers finish server-side training to perform aggregation. Similarly, edge servers do not wait till all the clients in their group finish client-side training to perform server-side training. The aggregation of the server-side model is performed once the number of client updates received by the edge server reaches $t\_threshold_g$. Therefore, $t\_threshold_g$ for each edge server could be dynamically adjusted to reduce excessive communication with the cloud server. To reduce the communication and computation overheads at the cloud server, a higher $t\_threshold_g$ should be chosen if there are more clients in group g or the clients in group g provide model updates more frequently. The exact $t\_threshold_g$ value is also dependent on the computation and communication capabilities of the cloud server, and can be determined by the negotiation between the edge and cloud servers.

Split layer selection: As shown in Fig. 5.1 and [108], [187], [188], the layers closer to the input layer generally contain more information related to the client's private dataset. If a client requires enhanced privacy, it should select a split layer closer to the output layer. However, the higher privacy comes at the cost of higher resource consumption since the client needs to train more layers. Because the client-side training only uses the private dataset from a single client, having more private layers could also lead to overfitting the client's private dataset and thus reduce the generality of the global model. Additionally, if a client shares fewer layers, the global model utility is lower as the cloud server receives fewer layers from the

Algorithm 6 Split-Aggregate Algorithm

Constants:

- Edge server: $\mathcal{E}_g$
- Client: $\mathcal{D}_{g,m}$
- Set of groups: G
- Set of clients in edge group g: N_g

Initialisation:

- Edge server models: $\mathbf{w}_g, g \in G$
- Client models: $\mathbf{w}_{g,m}, g \in G, m \in N_g$
- Client split layer: $\mathbf{l}_{g,m}, g \in G, m \in N_g$
- Aggregated cloud model: $\mathbf{w}_c$
- Aggregation counter: $t_g \leftarrow 0, g \in G$
- Aggregation time threshold: $t\_threshold_g, g \in G$

Input:

- Client dataset: $\mathbf{d}_{g,m}$ (inputs), $n_{g,m}$ (labels)

for $g\ in\ 1, 2, ..., |G|;\ m\ in\ 1, 2, ..., |N_g|$ **do in parallel**
(At client $\mathcal{D}_{g,m}$)
  $\mathbf{l}_{g,m} \leftarrow \texttt{ForwardPass}(\mathbf{w}_{g,m}, \mathbf{d}_{g,m})$
  $\texttt{SendtoEdge}(g, \mathbf{l}_{g,m}, n_{g,m})$
  $\mathbf{l}_{g,m} \leftarrow \texttt{ReceivefromEdge}(\mathbf{l}_{g,m})$
  $\texttt{BackwardPass}(\mathbf{w}_{g,m}, \mathbf{l}_{g,m})$
end for

for $g\ in\ 1, 2, ..., |G|$ **do in parallel**
(At edge server $\mathcal{E}_g$)
  $\mathbf{l}_{g,m}, n_{g,m} \leftarrow \texttt{ReceivefromClient}(\mathbf{l}_{g,m}, n_{g,m})$
  $\texttt{ForwardPass}(\mathbf{w}_g, \mathbf{l}_{g,m})$
  $\mathbf{l}_{g,m} \leftarrow \texttt{BackwardPass}(\mathbf{w}_g, n_{g,m})$
  $\texttt{SendtoClient}(g, m, \mathbf{l}_{g,m})$
  $t_g \leftarrow t_g + 1$
  if $t_g \geq t\_threshold_g$ **then** ▷ server-side aggregation
    $n_g \leftarrow \sum_{m=1}^{N_g} |n_{g,m}|$
    $\texttt{SendtoCloud}(\mathbf{w}_g, n_g)$
    $t_g \leftarrow 0$
    $\mathbf{w}_g \leftarrow \texttt{ReceivefromCloud}(\mathbf{w}_g)$
  end if
end for
(At cloud server)
$\texttt{ReceivefromEdge}(\mathbf{w}_g, n_g)$
$n \leftarrow \frac{n_g}{\sum_{g=1}^{G} n_g}$
for $l\ in\ 1, 2, ..., |\mathbf{w}_g|$ **do**
  $\mathbf{w}_c[l] \leftarrow n\mathbf{w}_g[l] + (1 - n)\mathbf{w}_c[l]$
end for

28: **for** $g\ in\ 1, 2, ..., |G|$ **do** ▷ broadcast model
29: $\mathbf{w}_g \leftarrow \mathbf{w}_c[0 : |\mathbf{w}_g|]$
30: $\texttt{SendtoEdge}_g(\mathbf{w}_g)$
31: **end for**

corresponding edge server. Such privacy-utility trade-off is further explored in Sec. 5.5 where some privacy measures could help clients to determine an optimised split layer.

Edge server selection: Because the split layer of the server-side models is determined by the edge servers and cannot be changed once determined, the clients need to select an edge server with a split layer of their interest, decided by the trade-off between privacy and utility. A client should select the closest edge server to reduce the communication cost if multiple edge servers have the same split layer. The edge servers belonging to the same organisation as the client should also be prioritised because they are more likely to share the same interests and less likely to behave maliciously toward the client.

Aggregation weight calculation: In Algorithm 6, the aggregation weights for the edge models are defined as n_g, which is proportional to the total size of the client datasets used in the training round. This is based on the assumption that all clients send their model weights on time and their connections are reliable. However, it should be noted that in a practical scenario, this assumption might not hold. Therefore, n_g can also be based on the combination of the size of clients' dataset, reputation level, or model performance. It is also assumed that the labels of the clients' dataset are shared with the edge server (lines 6 and 11). Similar to the conventional SL framework in [12], the clients might wish to perform backward propagation on the last layer to keep the labels of their dataset private. In this case, the clients only share the size of their dataset ($|n_{g,m}|$ in line 17) with the edge servers.

5.3 Theoretical Analysis of Efficiency and Fault Tolerance

In conventional FL [10], the cloud server performs model aggregation, and client devices perform model training. In *FlexSplit*, the number of models and layers aggregated at the cloud server is less compared to FL, as there are fewer edge servers than clients. This reduces the

communication and computation bottlenecks at the cloud server. Alternatively, in conventional SL [11], the cloud server performs part of the model training process sequentially whereas the cloud server in *FlexSplit* is only responsible for aggregating the server-side models which are trained by multiple edge servers in parallel. In *FlexSplit*, the clients can select the number of locally-trained layers, which is more efficient for large-scale heterogeneous systems where different client devices have various constraints in computation and communication resources.

Based on the hierarchy in Fig. 5.2, the communication complexity of *FlexSplit* can be estimated as

$$O\left(\sum_{g=1}^{G}\left(\left|\mathbf{w}_g\right|+\sum_{m=1}^{N_g}\left|L_{g,m}\right|\right)\right) \tag{5.1}$$

where $\left|\mathbf{w}_g\right|$ is the size of the server-side model sent to the cloud server for aggregation and $\left|L_{g,m}\right|$ is the size of the split layer transferred between clients and edge servers. The communication complexity of the cloud server is reduced in *FlexSplit* compared to the conventional FL because the server-side models are smaller than the entire model, and there are fewer edge servers than the clients. *FlexSplit* has higher communication complexity than SL due to the transfer of the server-side models. In SL, however, the communication between clients and the cloud server dominates the communication complexity. Whereas in *FlexSplit*, the communication is mainly between multiple pairs of a client and edge server, and between an edge server and the cloud server in parallel.

Because server-side model aggregation happens after multiple client updates and no model training is performed in the cloud server, it is difficult for the cloud server to reconstruct clients' private datasets using the edge-server model updates or hijack the model training by manipulating the server-side models. Alternatively, a client can benefit from other clients in the same edge group by server-side model training at the edge server, and benefit from clients in other edge groups by layer-wise model aggregation at the cloud server. The hierarchical structure and asynchronous layer-wise aggregation (lines 25-31, Algorithm 6) also reduce the single point of failure. For example, during an event of the cloud server or an edge server

drops out, or intermittent connection edge and cloud server, the system can still perform training and produce lower-performing models for the clients.

Compared with the SplitFed architecture proposed by Thapa et al. [189], the proposed *FlexSplit* technique does not require a third party to perform the aggregation which reduces the privacy risks and bottlenecks. Reducing the centrality and domination of the cloud server also reduces the privacy risks such as data reconstruction attacks caused by the cloud training [110].

Table 5.1 shows the comparison of the communication complexity of the entire system between *FlexSplit* and the existing FL, SL and SplitFed [189] architectures. For the existing solutions, it is assumed that the number of clients is equal to the total number of clients in all the groups in *FlexSplit*.

TABLE 5.1. Comparison of Communication Complexity between *FlexSplit*, FL, SL and SplitFed

	Communication Complexity
Proposed *FlexSplit*	$O\left(\sum_{g=1}^{G}\left(\left\|\mathbf{w}_g\right\| + \sum_{m=1}^{N_g}\left\|L_{g,m}\right\|\right)\right)$
Federated Learning [10], [190]	$O\left(\sum_{g=1}^{G}\sum_{m=1}^{N_g}\left\|\mathbf{w}_{g,m}\right\| + \left\|\mathbf{w}_g\right\|\right)$
SplitFed [189]	$O\left(\sum_{g=1}^{G}\sum_{m=1}^{N_g}\left(\left\|L_{g,m}\right\| + \left\|\mathbf{w}_g\right\|\right)\right)$
Split Learning [11]	$O\left(\sum_{g=1}^{G}\sum_{m=1}^{N_g}\left\|L_{g,m}\right\|\right)$

5.4 Simulation Setup

FEMNIST [180] – a preprocessed version of the Extended MNIST (EMNIST) [181] is considered as the benchmark dataset to simulate non-independent and identically distributed (non-IID) characteristics of clients' data. The dataset groups more than 805,000 images of hand-written characters by 3,500 different writers.

We simulate 50 clients divided into ten equally-sized edge groups. Each client uses ten writers for training and five unknown writers (i.e., writers not selected for training) for validation.

The task is to classify the characters into 62 classes of digits and letters. A commonly-used LeNet [191] CNN model is considered. It contains two convolution layers, each using a tanh activation function and followed by a maximum polling layer for the feature extraction module, and three fully-connected layers for the classification module. To prevent overfitting, the dropout rate between the feature extraction module and classification module is set to 0.25. Adam optimiser is utilised at a learning rate of $1e^{-4}$.

5.5 Simulation Results

This section presents the simulation results for the proposed *FlexSplit* technique.

5.5.1 Learning Performance

For SL, the last layer of the feature extraction module (i.e., layer four) is set to be the fixed split layer for all the clients. Alternatively, for *FlexSplit*, it is assumed that the majority of the clients wish to contribute more to the global model utility, whereas the minority of the clients require more privacy preservation. Therefore, among the ten groups of clients, the split layers for six groups are set to be layers one or two, while the other four groups are set to split at layers three to six. As shown in Fig. 5.3, the proposed *FlexSplit* with various split layers yields higher validation accuracy than SL with a fixed split layer because the *FlexSplit* server-side model is trained by multiple edge servers in parallel as opposed to a single cloud server sequentially. The layers closer to the input typically extract generalised features common to all the clients, whereas the latter layers extract information more specific to each client's dataset [119]. Therefore, *FlexSplit* outperforms centralised learning in terms of validation accuracy. However, the convergence of *FlexSplit* is slower than centralised learning as model training is partly distributed at different edge servers and clients, which introduces additional statistical heterogeneity during the training process. *FlexSplit* also provides higher validation accuracy and convergence rate than FL because part of the model training is decentralised at edge servers rather than fully distributed.

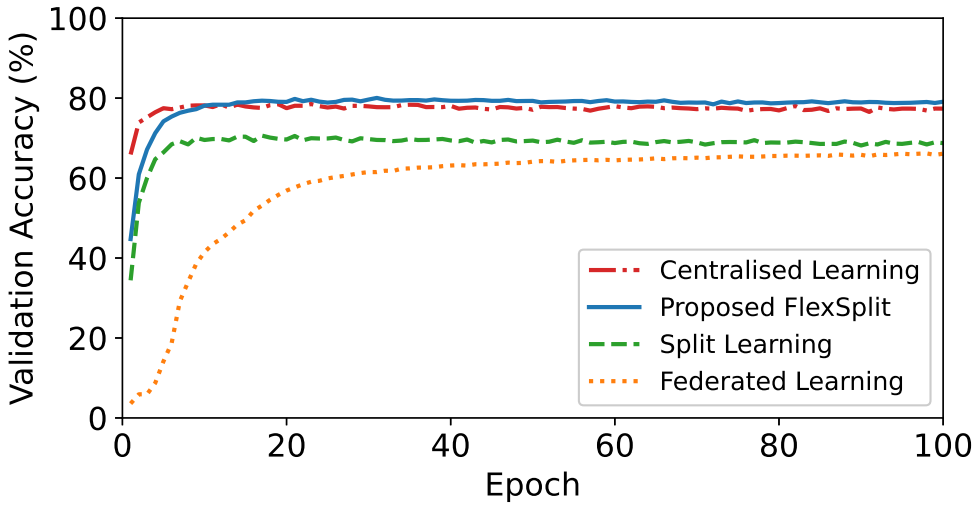


FIGURE 5.3. Benchmark Comparison of Model Accuracy Among Centralised Learning, FL, and SL at a Fixed Layer

5.5.2 Split Layers

Next, experiment results with different split layers based on the *FlexSplit* settings are presented. We compare the prediction accuracy of the training and validation datasets using the global model after 100 training rounds. The *FlexSplit* models with different split layers are compared with FL (i.e., split at the input layer) and centralised learning (i.e., split at the output layer). As shown in Fig. 5.4, when the split layer is closer to the output, the validation accuracy decreases and training accuracy increases, except for layer three which has lower validation accuracy than layers two and four. The accuracy for split layers one and two are similar to centralised learning, whereas the accuracy for split layer six is close to FL. In this figure, higher training accuracy with lower validation accuracy means that the model overfits the training set. Therefore, if a client selects a split layer closer to the output, its model tends to overfit their dataset. This is due to the increase in the portion of model training performed locally by clients using their private datasets, which are limited in the number of samples and statistical heterogeneity. The generalisation of the global model reduces as the reduction in the portion of layers at the server-side models contributed by all the clients. In general, all the clients should select a split layer as close to the input layer as possible to maximise their

contribution to the global model utility. Clients should consider this privacy-utility trade-off in choosing a privacy level to achieve the desired local model performance.

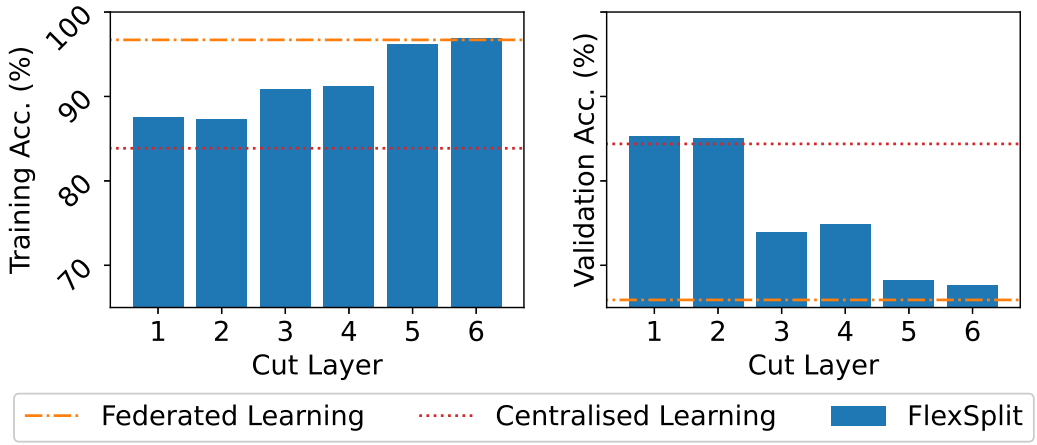


FIGURE 5.4. Model Utility and Generalisation at Different Split Layers.

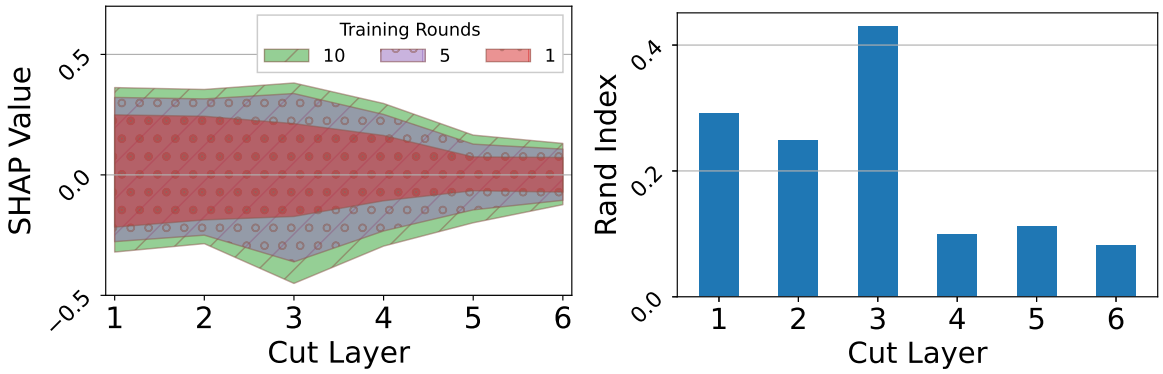
5.5.3 Privacy Preservation

To assess the effect of a client's privacy control on model utility, the split layer is adjusted for all the clients within the range between the first layer after the input (i.e., layer 1) and the last layer before the output (i.e., layer 6). The rest of the parameters remain unchanged. The SHapley Additive exPlanations (SHAP) technique [192] is utilised to determine the dependencies between the input features (e.g., image pixels) and the split layer output coefficients. In *FlexSplit*, specifically, the SHAP value can be used to measure the effectiveness of using machine learning-based algorithms to reconstruct the clients' private input features based on the clients' split layer outputs accessible by the edge servers (line 11, Algorithm 6). The SHAP values for all the input features with respect to the activation outputs from different split layers are calculated for the models after one, five, and ten training rounds.

SHAP library[1] written in Python is utilised to compute the SHAP values and the code is modified to use the split layer as the prediction output instead of the original output layer. As such, the dependency between each input feature and split layer activation outputs would be determined. The attacker can reverse the process and determine the input features based on

[1]https://shap-lrjball.readthedocs.io/

the split layer activation outputs it obtains from an edge server or during the communication between a client and an edge server.



(A) The range of the SHAP value indicates how much the input features contribute towards the change of the outputs at different split layers. A value closer to zero corresponds to higher privacy preservation of the client's data. The maximum possible range is [-1,1] where negative and positive values represent negative and positive correction, respectively. Zero means there is no dependency.

(B) Clustering Rand indexes indicate how well the clustering model separates the split layer outputs from different clients. The Rand Index ranges between [0,1] where 0 means there is no similarity between correct and predicted clusters, and 1 means all the clusters are separated correctly.

FIGURE 5.5. Privacy Preservation at Different Split Layers

Fig. 5.5a shows the range of all the SHAP values for different parameters. We can see that the range of the SHAP value generally decreases which corresponds to a lower privacy leakage for the clients when the split layer is closer to the output layer and the number of training rounds is smaller. The figure also shows that the SHAP value slightly increases at split layer three after five or ten training rounds. This is because layer three is the second last layer of the feature extraction module where the convolution layer may have extracted some important input features that are useful for performing reconstruction attacks.

Next, the feasibility of determining the source of the client models is considered given the split layer. It is assumed that the clients send the model updates anonymously and their identities are unknown to any external entity. The attackers aim to separate the models from different clients and predict the identity of the client to whom a future model belongs. The attackers can perform unsupervised K-means clustering on the split layer outputs to separate

them based on the distance between them and utilise some auxiliary data like IP addresses to reveal client identities.

Training of the K-means clustering model is performed using the split layer outputs from the first 100 training rounds, and use the trained model to cluster the split layer outputs for the following 100 training rounds. We use the Rand Index, which is calculated by dividing the number of correctly separated pairs of samples belonging to different or the same clusters by the total number of pairs. Fig. 5.5b shows that if a client moves the split layer closer to the output layer, the Rand Index of the clustering model generally reduces. Similar to Fig. 5.5a, split layer three is an exception that has the highest Rand Index. This means that split layer three also extracts some important features about the client's identity. The attacker can train a K-means clustering model using 100 outputs from split layer three and correctly separate more than 40% of the split layer outputs from different clients. Comparatively, the success rate of K-means clustering is reduced to around 10% if a client selects the split layer at layers four, five, or six.

5.5.4 Discussion

From the simulation results for the FEMNIST dataset, the clients should avoid split layer three as it provides lower model utility and the least protection against privacy attacks simulated. The client can decide on their split layer based on the privacy-utility trade-off. Apart from split layer three, user privacy can be better preserved if a client considers a split layer closer to the output layer. However, privacy preservation comes at the cost of more resource consumption for client-side model training and less model utility due to the lack of collaboration between clients and groups.

5.6 Summary

This chapter proposed a personalised privacy preservation technique for DML named *FlexSplit*. *FlexSplit* leverages the SL architecture with multiple edge servers splitting at different layers of the ANN model. The clients in *FlexSplit* are allowed to select an edge server that meets

their privacy and security requirements. Theoretical analyses highlighted the advantage of *FlexSplit* over conventional FL, SL and SplitFed frameworks in terms of computational and communications efficiencies. Finally, simulation results showed the trade-offs between privacy, model utility and resource consumption that the clients should consider when selecting their split layer.

CHAPTER 6

Personalised Masking for Privacy-Preserving Distributed Machine Learning

---

6.1 Introduction

Chapter 5 presented a flexible hybrid FL-SL structure that allows each client to select its preferred split layer based on their privacy level. The proposed *FlexSplit* architecture is based on the conventional FL and SL approaches with an additional hierarchical level for different clients to select their preferred edge servers with different split layers. This chapter presents an alternative approach where the SL clients apply masks to the activation outputs of neurons in their client-side models to actively protect their privacy.

A more advanced PIA adversarial model is adopted in this chapter compared to the DRA adversarial model considered in Chapter 5. PIA is a type of attack where an adversary trains an auxiliary classifier aiming to infer properties of the input features (i.e., hidden attributes) that are unrelated to the output labels. The conventional white-box PIA [102], [193]–[196] assumes that all the model parameters are exposed to the adversary. Under such a white-box assumption, the adversary has full access to the model. However, this assumption is unlikely to hold in practical SL scenarios, as the client-side models in SL remain private and the model parameters are not shared with any external agent in the system. More recent studies on black-box PIA weakened this assumption such that the adversary is only able to query the model and receive the activation outputs from the neurons in the model [110], [197], [198]. Some of the black-box attacks also assume that the adversary can have control of the model training process to some degree by injecting poisoned data or hijacking the training process [110], [195], [196], [198]. However, the SL clients have full control of the client-side

model and input features. Therefore, the black-box PIA attacks cannot be performed in practice as the assumptions are still too strong for the SL scenario.

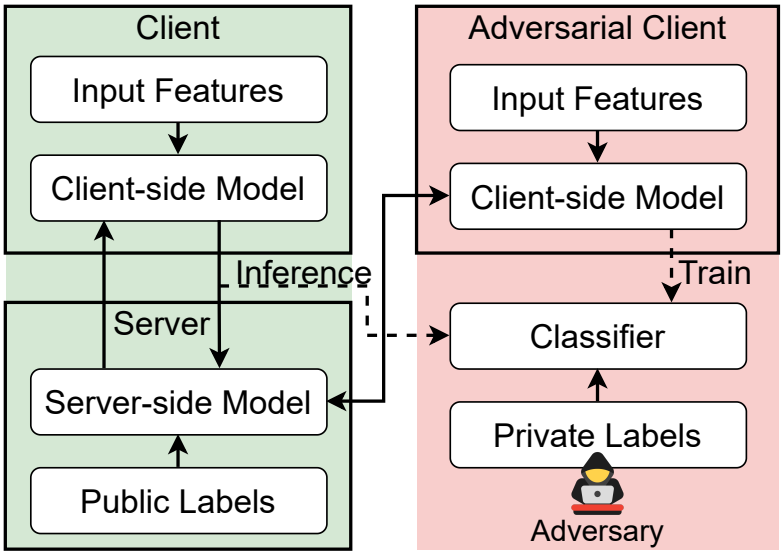


FIGURE 6.1. Adversarial Model for PIA in SL without Privacy Preservation

Therefore, this chapter further investigates a PIA adversarial model shown in Fig. 6.1. In Fig. 6.1, the green area represents the typical SL training process, and the red area represents the adversary process. Different from the conventional black-box and white-box attacks, the adversary in Fig. 6.1 does not have any control over the honest client's model and it cannot inject any data into the honest client's model.

This chapter also proposes a personalised masking technique named *FlexMask* to defend against such an attack by applying a mask to the client-side model. The *FlexMask* technique proposed in this chapter allows clients to apply masks to the activation outputs of neurons in their client-side models based on the private attributes they wish to protect. The sensitivities of the neurons to the selected private attribute are measured by similarity metrics, and the portion of the masked neurons can be adjusted based on the trade-off between clients' privacy levels and global model utility.

6.2 Adversarial Model

It is assumed that the adversary trains its client-side model to behave similarly to a typical honest client-side model. Therefore, as shown in Fig. 6.1, an honest but curious adversary pretends to be a typical client training the model using a legitimate dataset (i.e., the public dataset) without poisoning the model [1]. However, it aims to infer private attributes of other clients' private datasets by training an auxiliary binary classifier using the activation outputs from the adversarial client-side model and their corresponding private attributes.

Non-adversarial clients prefer to keep private attributes of their dataset confidential and share only the labels of the original learning task (i.e., public labels) with the server. The private attributes that the non-adversarial clients aim to protect must be different from the original learning task, and they are unknown to the adversarial client. However, they are unaware of the adversarial client's presence and the private attributes that the adversary is interested in inferring.

The adversarial clients may be involved at any time throughout the training and inference process. The trained auxiliary classifier is then used to infer the private attributes of the private dataset based on the activation outputs from non-adversarial clients' models. We define a sample with a *positive label* as one that possesses the attribute in the data sample, while a sample with a *negative label* lacks this attribute.

6.3 The Proposed FlexMask Technique

This Section details the configuration of the proposed *FlexMask* technique. It begins with an overview of the technique and then explores the details in the following subsections.

[1] While we assume that the adversary trains its model and behaves similarly to a typical non-adversarial client, the adversary may perform harmful actions to the model (e.g., by injecting poisoned data or model parameters). However, this comes at a cost that the adversarial client-side model behaves differently from the non-adversarial client-side models, which is harmful to the adversary itself. Therefore, it is assumed that the adversary's intention is to infer clients' private attributes without poisoning their model.

6.3.1 Overview of FlexMask

Based on the theory that different neurons in an ANN model are responsible for different semantic labels [199]. By masking those sensitive neurons, a client can prevent adversaries from inferring the private attributes. Therefore, a client first needs to determine the sensitivities of the neurons in different layers to the private attribute it wishes to hide. Once determined, the client needs to decide what subset of the highly sensitive neurons to mask and how to mask those neurons while balancing privacy and model utility trade-off. Therefore, the proposed technique aims to address the following research questions (RQs) to achieve an acceptable privacy-utility trade-off as determined by the client:

- **RQ1:** Which neurons in the chosen layer are good candidates for masking?
- **RQ2:** What subset of those candidate neurons should be masked?
- **RQ3:** How should a chosen neuron be masked?
- **RQ4:** When should the mask be applied?
- **RQ5:** Which layer should be masked?

These questions are interrelated and should be answered together to arrive at an optimum privacy-utility trade-off. However, such a combinatoric optimisation problem is NP-hard. Hence, a heuristic approach is adopted, aiming to answer one question at a time.

In *FlexMask*, the client determines a personalised mask based on the activation outputs from the neurons in the client-side model, as well as the client's labels of private attribute (i.e., private labels) and privacy level. Given that private labels are binary, we propose to determine the sensitivities of neurons by examining how they respond to positive and negative private labels. For this, we leverage similarity measurements that calculate the information shared between two distributions of neurons' activation outputs corresponding to samples with positive and negative labels of the same private attribute. A high similarity score indicates that the neuron behaves very similarly for samples with positive and negative private labels. Conversely, the lower the similarity score, the less information is shared between the two sets of neurons' activation outputs with positive and negative private labels. Therefore, the neuron is more sensitive to the private attribute.

Fig. 6.2 illustrates the proposed *FlexMask* mask determination procedure at a client. First, the client determines the sensitivities of the neurons to the private attributes it wishes to protect. This can be accomplished by applying a separation metric that measures the similarity of the client-side model's activation outputs for each neuron corresponding to positive and negative private attributes. Second, the client determines the neurons to mask based on its desired privacy level and model utility requirement. Analysis in Sec.6.5 shows that masking a fraction of 15-25% of neurons gives a good privacy-utility trade-off. Then, the activation outputs from neurons with lower similarity scores are replaced as they are the ones most sensitive to the attribute the client wishes to keep private.

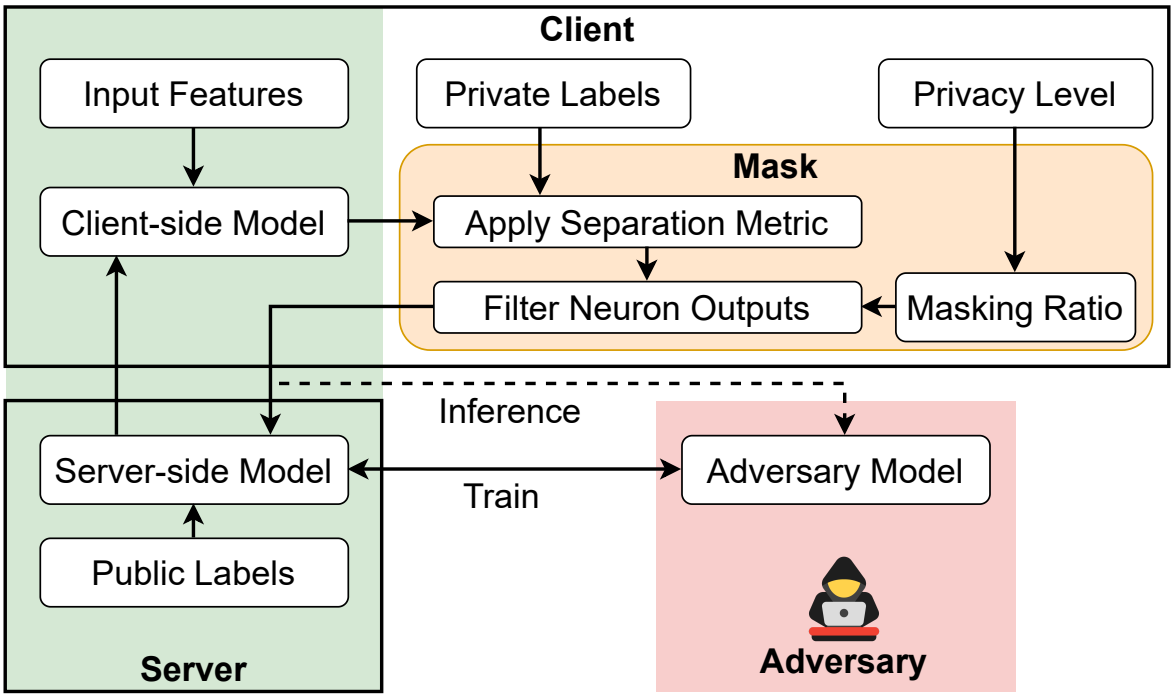


FIGURE 6.2. *FlexMask* Mask Determination and Application Procedures at a Client

Fig. 6.3 illustrates the application of masks. Dark-coloured neurons represent the candidates for masking as identified by the above mask determination technique. The blue squares indicate that the activation outputs of these neurons are masked using the above mask application technique, where their activation outputs are replaced with any values before being connected to the next layer. This masking procedure can be applied to any layer of the client-side model.

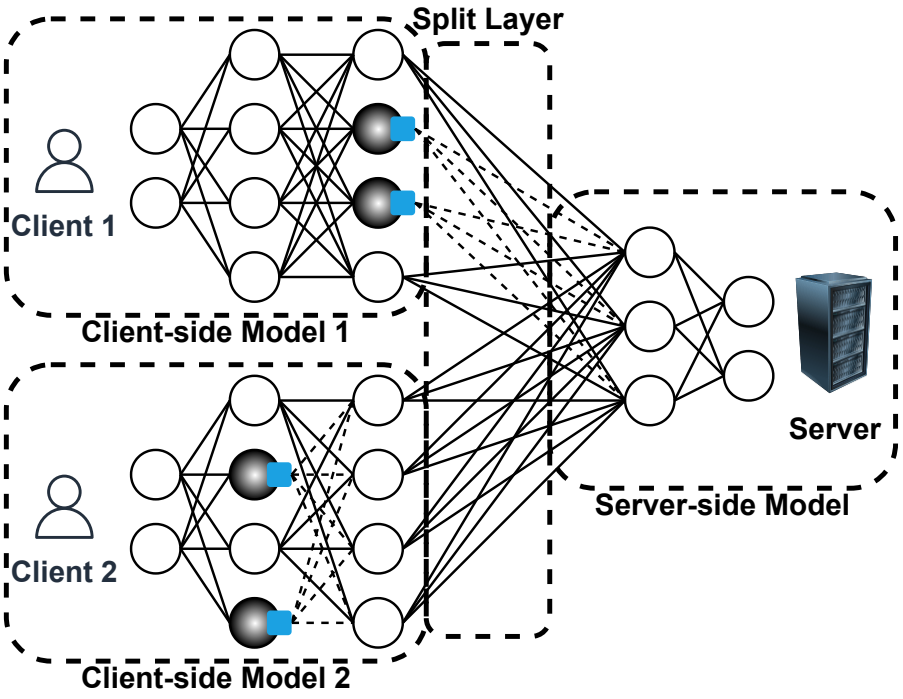


FIGURE 6.3. Neuron Masking with FlexMask

To avoid information leakage, each client locally determines the masks based on its client-side model's mask layer activation outputs and their corresponding private attributes. Ideally, if there is no overlap between the set of neurons sensitive to the private and the public attributes, then those neurons can be masked to achieve perfect protection without affecting the model utility. However, this is unlikely to be achieved in practice. The goal of masking is to separate the neurons as accurately as possible such that the neurons that are more sensitive to the private attribute are masked with minimal impact on the utility of the original model.

6.3.2 Determining Neurons to Mask

The layer where the mask is applied is defined as the *mask layer*. The mask layer must be a layer within the client-side model. Let $\mathbf{c}_m$ be the set of mask layer activation outputs at client m:

$$\mathbf{c}_m = \{\mathbf{c}_{m,1}, \ldots, \mathbf{c}_{m,d}, \ldots, \mathbf{c}_{m,n_m}\} \tag{6.1}$$

where d is the data sample index, and m_n is the total number of data samples provided by client m in a particular training round. Let $\mathbf{c}_{m,d}$ be the set of activation outputs from all mask layer neurons of client m for data sample d:

$$\mathbf{c}_{m,d} = \{c_{m,d,1}, \ldots, c_{m,d,r}, \ldots, c_{m,d,R}\} \tag{6.2}$$

where r is the neuron index and R is the total number of neurons in the mask layer.

Next, the public and private labels for client m are denoted as follows:

$$\mathbf{y\_pub}_m = \{y\_pub_{m,1}, \ldots, y\_pub_{m,d}, \ldots, y\_pub_{m,n_m}\} \tag{6.3}$$

$$\mathbf{y\_pvt}_m = \{y\_pvt_{m,1}, \ldots, y\_pvt_{m,d}, \ldots, y\_pvt_{m,n_m}\} \tag{6.4}$$

A client first separates the vectors of mask layer activation outputs according to the private attributes intended to be protected using Algorithm 7.

Algorithm 7 Function for Separating a Client's Split Layer Activation Outputs According to the Private Attributes

Input:

- Split layer activation outputs from client m: $\mathbf{c}_m$
- Private attributes for client m: $\mathbf{y\_pvt}_m$

Output:

- Split layer activation outputs corresponding to samples with positive and negative private attributes for client m: $\mathbf{c\_pos}_m$ and $\mathbf{c\_neg}_m$

function CLIENTACTSEP($\mathbf{c}_m, \mathbf{y\_pvt}_m$)
  $\mathbf{c\_pos} \leftarrow \{\}$
  $\mathbf{c\_neg} \leftarrow \{\}$
  for d *in* $1, ..., n_m$ **do** ▷ loop through all samples
    if $y\_pvt_{m,d}$ is positive **then**
      $\mathbf{c\_pos}$.`insert`($\mathbf{c}_{m,d}$)
    else
      $\mathbf{c\_neg}$.`insert`($\mathbf{c}_{m,d}$)
    end if
  end for
  return ($\mathbf{c\_pos}, \mathbf{c\_neg}$)
end function

Then, in Algorithm 8, the mask the sensitivities of the neurons in the mask layer are determined based on a similarity metric $\texttt{sim}(\textbf{c\_pos}, \textbf{c\_neg})$. It is assumed that at least one sample has positive and negative private labels. If all the samples belong to the same class, the client might use samples from a public dataset for the other class or generate synthetic data for the other class using techniques like GAN [200].

In Algorithm 8, the similarity scores of the neurons $r \in [1, R]$ in the mask layer are calculated. The $\texttt{sort}(\textbf{sim\_c})$ function is then used to sort the similarity scores for all neurons in the mask layer in ascending order. A similarity score $threshold$ is used to determine the subset of neurons to mask, which is the product of the masking ratio p_mask and the total number of neurons in the mask layer R. Thereby, the portion of the masked neuron is p_mask. Then, the neurons' similarity score is traversed in the original order before sorting. If a neuron's similarity score is less than the threshold, it is considered sensitive to the private attribute and therefore masked. We set the labels of the masked neurons as 1, whereas the unmasked neurons are labelled 0. The determined mask label vector for client m is denoted as $\mathbf{l}_m$.

Given a data sample, the mask layer activation outputs could be changed dynamically for different training rounds. Because the client devices might be resource-constrained, the chosen measurement metric needs to be calculated with minimal computational overhead. This is to prevent excessive delays in transmitting the split layer activation outputs to the server. After testing a range of existing similarity measurement metrics, we consider the following metrics with lower computational overheads as examples:

(1) **Centroid (Cen):** In the basic centroid-based distance metric, centroids of a mask-layer neuron's activation outputs corresponding to samples with positive and negative private labels are calculated as the mean values of the two sets of activation outputs: **c_pos** and **c_neg**. The distance between centroids would be the absolute value of the difference between the two centroids. We consider the neurons with larger distances to have greater dissimilarity. Therefore, the neurons with larger distances are more sensitive to the private attribute.

(2) **Correlation (Cor):** Pearson correlation coefficient [201] measures the linear correlation between two variables. We consider the correlation coefficient between the

Algorithm 8 Function for Determining the Mask for Clients Based on Activation Outputs Separated According to Private Attributes

Input:

- Mask layer activation outputs correspond to positive and negative private attributes for client m: **c_pos**$_m$ and **c_neg**$_m$
- Fraction of neurons to be masked: p_mask

Output:

- Vector of labels for whether a neuron should be masked (1) or unmasked (0) for client m: **l**

function CLIENTMASK(**c_pos**$_m$, **c_neg**$_m$, p_mask)
  sim_c $\leftarrow \{\}$
  l $\leftarrow \{\}$
  for r *in* $1, ..., R$ **do** ▷ loop through all neurons
    c_p $\leftarrow \{c[r] \; \forall \; c \in$ **c_pos**$_m\}$
    c_n $\leftarrow \{c[r] \; \forall \; c \in$ **c_neg**$_m\}$
    sim_c.insert(sim(**c_pos**, **c_neg**)) ▷ distance
  end for
  sorted_dist $\leftarrow$ sort(**sim_c**)
  $threshold \leftarrow$ **sorted_sim**$[R \times p\_mask]$
  for r *in* $1, ..., R$ **do** ▷ go through all neurons
    if sim_c$[r] < threshold$ **then**
      l.insert(0)
    else
      l.insert(1)
    end if
  end for
  return l
end function

activation outputs from a neuron and their corresponding samples' private labels (-1 as negative and one as positive). The absolute value of the Pearson correlation is used as the similarity measurement because the activation outputs can be either positively or negatively correlated to the private labels. The results should be ranged between zero and one, where a larger value indicates that the neuron has higher sensitivity to the private attributes because it is correlated to the private label.

(3) **Kullback-Leibler Divergence (KLD):** Kullback-Leibler divergence [202] is widely used to measure the shared information between two probability distributions. Due to the limited number of discrete samples, we utilise the k-nearest-neighbour density estimation method [203], which approximates the KL-divergence value based on the

empirical cumulative distribution function (eCDF). A KL-divergence value of zero indicates that the two distributions are identical, meaning that the neuron outputs similarly for positive and negative private labels. Whereas a larger value indicates a high dissimilarity between the two distributions, which suggests that the neuron has higher sensitivity to the private attribute.

(4) **Calinski-Harabasz Score (CHS):** The CHS [204] (also known as Variance Ratio Criterion) measures the dispersion of two clusters. It is calculated by dividing the sum of between-cluster dispersion by the sum of within-cluster dispersion. A higher CHS value indicates better separation between clusters and denser samples within the clusters, which suggests higher sensitivity of the activation outputs according to their corresponding private labels.

(5) **Davies-Bouldin Score (DBS):** The DBS [205] measures the similarity between two clusters. It is a distance-based metric calculated by dividing the summation of the average distance between each sample and the centroid of its corresponding cluster by the distance between cluster centroids. A lower DBS value typically indicates lower similarity between clusters, which suggests higher sensitivity of the activation outputs according to their corresponding private labels.

(6) **Random Masking (Rand):** If a client is unable to find a dataset with reliable private attributes or lacks the computing capacity to ensure timely updates of the masks, they can randomly mask the activation outputs of the mask layer. This is similar to the conventional random dropout technique [140].

6.3.3 Applying Mask to Neurons Activation Outputs

The mask label vector $\mathbf{l}_m$ is determined during the forward propagation of the client-side model using Algorithm 8. After determining $\mathbf{l}_m$, the mask should be applied to the mask layer activation outputs during the forward propagation process. Because the structure of the server-side model is unchanged, the masked neurons' activation outputs need to be replaced by some values that are irrelevant to the original activation outputs but within a similar range to

maximise privacy protection and preserve model utility. The following two mask application strategies are considered:

(1) **Constant:** Replacing masked neurons' activation outputs with a constant number such as zero (i.e., deactivating the masked neurons). This method requires less computing power because only the matrix multiplication and addition operations are required to be executed once to apply the mask to the activation outputs. However, this may signal to the attacker that the client is attempting to hide certain properties. It may even reveal the property the client intends to protect, as the server can easily determine the most frequent value in the activation outputs to infer the patterns of the masked neurons.

(2) **Random:** Replacing masked neurons' activation outputs with randomly generated values within a similar range of the original activation outputs from all neurons in the mask layer. It is assumed that the neurons' activation outputs follow Gaussian distributions. Let the mean value of all the activation outputs from client $\mathcal{D}_m$'s model during a training round be μ_m, and their standard deviation be σ_m. Then masked client $\mathcal{D}_m$'s activation outputs are sampled from the distribution $\mathcal{N}(\mu_m, \sigma_m^2)$. Random masking has better privacy compared to the constant masking strategy, as the attacker cannot use the distribution of activation outputs to infer the pattern of the masked neurons. However, enhanced privacy comes at the cost of increased computational cost due to the additional random sampling process.

(3) **Hybrid Constant+Random:** Combining constant and random strategies by using pre-calculated constant random numbers without resampling them or resampling periodically. The hybrid method is not considered in the experiments because the privacy protection effectiveness and model utility are expected to be between the two extreme case scenarios.

6.3.4 Model Training with FlexMask Technique

As described in Sec. 6.3.2, a client first separates mask layer activation outputs based on their corresponding private attributes using the function `ClientActSep`($\mathbf{c}_m, \mathbf{y\_pvt}_m$) in

Algorithm 7. The function outputs $\mathbf{c\_pos}_m$ and $\mathbf{c\_neg}_m$, which are two sets of mask layer activation outputs correspond to the data samples with positive and negative private labels, accordingly. Then, the client determines its personalised mask label vector $\mathbf{l}_m$ using the function `ClientMask`($\mathbf{c\_pos}_m, \mathbf{c\_neg}_m, p\_mask$) in Algorithm 8 based on a selected similarity measurement.

A training round for client m is then defined in Algorithm 9, where the set of input features for client $\mathcal{D}_m$ is denoted as $\mathbf{f}_m$. Lines 3-7 and 12-13 are executed at client $\mathcal{D}_m$ and lines 8-11 are executed at the cloud server. `ClientForward` and `ServerForward` are functions that perform forward propagation on the client- and server-side models, which return the activation outputs from the client- and server-side models, correspondingly. `ClientBackward` and `ServerBackward` are functions to perform backward propagation on the client- and server-side models, which update the parameters of the client- and server-side models, respectively. `ApplyMask` re-performs forward propagation on the client-side model with the binary mask applied to the mask layer activation outputs using one of the methods described in Sec. 6.3.3. The split layer activation outputs after processing are then transmitted to the server for training the server-side model.

6.4 Experimental Setup

A range of experiments are performed to determine the effectiveness of the proposed mask determination and application techniques, as well as masking ratios for private attributes with different properties. GPU is used to accelerate training and run multiple tasks simultaneously to maximise efficiency. A server equipped with i9-12900 CPU and NVIDIA RTX A2000 GPU is used. The memory size is 128 GB. The main ANN model and the adversarial model are based on the Pytorch[2] and scikit-learn (sklearn)[3] Python libraries, respectively.

[2]https://pytorch.org/
[3]https://scikit-learn.org/

Algorithm 9 Function for Training a Model with *FlexMask* Privacy Preservation

Input:

- Input features for client m: $\mathbf{f}_m$
- private attributes for client m: $\mathbf{y\_pvt}_m$
- Fraction of neurons to be masked: p_mask

function TRAINING($\mathbf{f}_m, \mathbf{y\_pvt}_m, p\_mask$)
At client $\mathcal{D}_m$:
  $\mathbf{c}_m \leftarrow \texttt{ClientForward}(\mathbf{f}_m)$
  $\mathbf{c\_pos}_m, \mathbf{c\_neg}_m \leftarrow \texttt{ClientActSep}(\mathbf{c}_m, \mathbf{y\_pvt}_m)$
  $\mathbf{l} \leftarrow \texttt{ClientMask}(\mathbf{c\_pos}_m, \mathbf{c\_neg}_m, p\_mask)$
  $\mathbf{cm}_m \leftarrow \texttt{ApplyMask}(\mathbf{c}_m, \mathbf{l})$
  $\texttt{TransmitToServer}(\mathbf{cm}_m)$
    ▷ transmit the masked activation outputs to the server
At server s:
  $\mathbf{cm}_m \leftarrow \texttt{ReceiveFromClient}(m)$
  $\mathbf{o}_s \leftarrow \texttt{ServerForward}(\mathbf{cm}_m)$
  $\mathbf{c}_s \leftarrow \texttt{ServerBackward}(\mathbf{o}_s, \mathbf{y}_m)$
  $\texttt{TransmitToClient}(\mathbf{c}_s, m)$
    ▷ transmit the split layer gradients to client m
At client $\mathcal{D}_m$:
  $\mathbf{c}_s \leftarrow \texttt{ReceiveFromServer}(s)$
  $\texttt{ClientBackward}(\mathbf{c}_s)$
end function

6.4.1 Dataset

CelebFaces Attributes (CelebA) dataset is an image dataset that contains 202,599 face images from 10,177 celebrities. Each image is annotated with 40 binary facial attribute labels. To simulate the limitations in data samples in real-life scenarios, images from 20 random celebrities are selected to form the public dataset and images from 30 celebrities to form the private dataset for training. The public dataset contains approximately 900 images, whereas the private dataset contains approximately 1350 images. The validation dataset contains images from 100 celebrities, which is approximately 4500 images.

Attributes: Table 6.1 lists the public attributes considered for the simulation and their corresponding classification tasks. The labels of the public attribute of all data samples are used as the public labels. 13 attributes from the CelebA dataset, listed in Table 6.2, are selected as the private attributes. The labels of the private attributes for all data samples are

used as the private labels. The remaining 27 attributes in the dataset are not selected for the following reasons:

- Already a public attribute: male, smiling, and young
- Non-binary attributes: black, blond, brown, grey, straight, and wavy hair
- Colour-related attributes (e.g., could be influenced by lighting conditions): pale skin and rosy cheeks
- Facial hairstyles (typically correlated to the gender): five o'clock shadow, goatee, mustache, no beard, and sideburns
- Non-facial attributes: attractive, bangs, blurry, bushy eyebrows, heavy makeup, mouth slightly open, wearing earrings, wearing hat, wearing lipstick, wearing necklace, and wearing necktie

TABLE 6.1. Portions of Positive Samples of the Public Attributes and Their Corresponding Classification Tasks

Attribute	Classification Task	% Positive Samples
Male	Gender classification	41.63
Smiling	Emotion classification	48.31
Young	Age classification	77.74

TABLE 6.2. Portions of Positive Samples of the Private Attributes

Attribute	% Positive Samples	Attribute	% Positive Samples
arched eyebrows	26.90	eyeglasses	6.42
bags under eyes	20.48	high cheekbones	45.60
bald	2.24	narrow eyes	11.58
big lips	24.24	oval face	28.28
big nose	23.42	pointy nose	27.86
chubby	5.76	receding hairline	7.84
double chin	4.70		

Data Distribution: As shown in Table 6.2, most of the selected private attributes are highly imbalanced in the dataset. For example, the portions of positive samples with private attributes bald, chubby, double chin, eyeglasses, and receding hairline are less than 10%. It is assumed that the honest clients' private datasets have relatively balanced private labels by selecting an equal number of celebrities, with a majority of the samples having positive and negative

private labels. However, the distribution of the private labels might not be known to the adversary. Therefore, the experiments are repeated under two different data distribution assumptions:

(1) **Balanced public dataset:** assuming the public dataset has similar numbers of samples with positive and negative private labels. An equal number of celebrities are selected randomly, with the majority of the samples being positive private labels and negative private labels.
(2) **Imbalanced public dataset:** assuming the public dataset is randomly selected from the entire pool of celebrities. The public dataset is therefore expected to follow a similar distribution of the private attribute to the original dataset in Table 6.2, which is different from the distribution of the honest clients' private datasets.

6.4.2 Model

For the experimental ANN model, a standard LeNet-5 model structure [19] is adopted to simulate training in a resource-constrained environment. The model has two pairs of convolutional and max pooling layers connected in serial, followed by three fully-connected layers. The rectified linear unit (ReLU) function is used as the activation function after each convolutional and fully-connected layer. Adam optimiser with a learning rate of 0.0005 is used, and cross-entropy loss is used as the loss function. To verify the model performance, trial training rounds are performed using 1342 samples from 30 randomly selected clients. The original validation accuracies are 84.65%, 81.27%, and 78.81% after 100 training rounds for 'male', 'smiling' and 'young' pubic attributes, respectively.

For the adversarial model, a binary classification model should be selected. After comparing k-nearest neighbours, support vector machine, decision tree, ensemble learning, and SLP classifiers in the sklearn library, it is observed that the SLP binary classifier generally performs better than others. Using different hyper-parameters does not have a significant improvement in the classification performance compared to using the default hyper-parameters (i.e., a single

hidden layer with 100 neurons). Therefore, we use the SLP binary classifier with default hyper-parameters for the adversarial model for all attack simulations for a fair comparison.

6.4.3 Splitting and Masking Techniques

In Sec. 6.3, the techniques for mask determination and application are proposed. The experiments will consider the following variants of splitting and masking techniques listed in Sec. 6.3.1.

Determining the mask (RQ1 and RQ2): The similarity metrics listed in Sec. 6.3.2, namely Cen, Cor, KLD, CHS, DBS, and Rand, are considered to determine the sensitivity of neurons in the mask layer. For each similarity metric, the experiments are repeated while varying the masking ratio p_mask to mask 5%, 15%, 25%, 50%, and 75% of the most sensitive neurons in the mask layer.

Applying the mask to the neurons (RQ3 and RQ4): The mask application methods listed in Sec. 6.3.3 are considered, including:

- Constant: consistently set the activation outputs of the masked neurons to 0; and
- Random: set the activation outputs of the masked neurons to random numbers sampled from a Gaussian distribution $\mathcal{N}(\mu_m, \sigma_m^2)$.

To simulate a scenario where the clients are limited in computing resources, the client can choose to mask at chosen training rounds to maximise the effectiveness of masking with minimal resource overheads. For attack and defence performance simulations, a worst-case scenario is considered where the client can only apply the mask once. Initial trial runs show that the validation accuracy of the model reaches the maximum at approximately 100 training rounds, which suggests that the model converges at approximately 100 training rounds. Therefore, the number of training rounds before generating and applying the mask is altered within the range between 1 and 100 to compare the attack protection effectiveness before and after the convergence of the model.

To analyse the impact on model utility, a worst-case scenario is also considered in which the clients always apply the mask on the mask layer during training. This is expected to negatively impact the model utility more than applying masks during a subset of the training rounds. Masked neurons' pattern is updated once every ten training rounds because the similarity measurements tend to be consistent across consecutive training rounds.

Split and mask layers (RQ5): The experiments consider all possible split layers and mask layers in the LeNet-5 CNN, including:

- Convolutional (C) layers: the first and second convolutional layers, with and without applying the ReLU activation function;
- Max pooling (MP) layers: the max pooling layers attached to the first and second convolutional layers; and
- Fully-connected (FC) layers: the first and second fully-connected layers, with and without applying the ReLU activation function.

6.4.4 Performance Measurements

Multiple metrics are considered to evaluate the effectiveness of privacy preservation and model the utility of the proposed *FlexMask* technique.

Privacy preservation: The attack success rate (ASR) is the adversarial classification model's validation accuracy for the target private attribute in percentage. Let $ASR_{original}$ represent the original ASR without masking, and ASR_{masked} represent the ASR of the masked model. The change in ASR is defined as:

$$\Delta ASR = ASR_{original} - ASR_{masked} \tag{6.5}$$

A positive ΔASR indicates that masking reduces the ASR compared to the original model, thereby enhancing privacy. A larger ΔASR signifies a higher level of privacy. Conversely, a negative ΔASR suggests that masking is ineffective in protecting privacy. For binary private

labels, 50% ASR is similar to a random guess, as only half of the inference results for the private attribute are correct.

Model utility: Validation accuracy (VA) is defined as the ANN classification model's accuracy in inferencing the public attributes of the validation dataset in percentage. Let $VA_{original}$ represents the original VA without masking, and VA_{masked} represents the VA of the masked model. The change in VA is then defined as:

$$\Delta VA = VA_{original} - VA_{masked} \tag{6.6}$$

A larger value of ΔVA signifies a negative impact on the model utility.

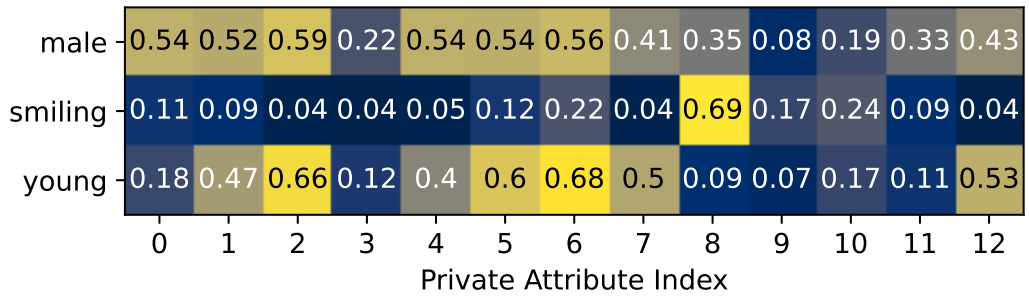
Computation costs: The computation costs for *FlexMask* consist of two parts: mask generation time and training time overheads. Mask generation time is the time taken to execute the mask determination algorithm from Sec. 6.3.2. For random masking, mask generation time is the time to generate masked neuron indexes randomly. The training time overhead is the time taken to apply the mask to the determined masked neurons at the mask layer. The masking algorithm is executed for 100 epochs and the average computation time overheads for a single epoch are used as the results.

6.5 Performance Analysis

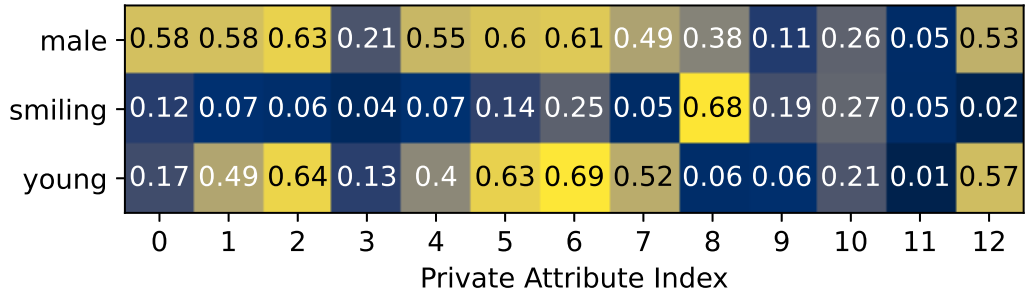
After conducting some initial trial runs, it is observed that when the original ASR is less than or close to 50%, masking is unnecessary as the model is already difficult to attack without applying a mask. As discussed in Sec. 6.4.4, those cases are considered unsuccessful attacks for binary private attributes. For the VA results, the original VA is used as the baselines, which are 84.65%, 81.27%, and 78.81% for 'male', 'smiling' and 'young' public attributes, accordingly.

6.5.1 Effectiveness of FlexMask for the Relationship between Private and Public Attributes

Because the clients are unaware of the adversary's models and datasets, the effectiveness of PIA and its ASR are unknown to them. Therefore, a client needs a means to estimate the ASR based on the known properties of their private datasets. After exploring various measurements, it is observed that the MI [206] between the public and the private attributes is related to the original ASR and ΔASR. A higher MI indicates that the two attributes share more information, which means that they are more similar. Whereas a lower MI indicates a weaker relationship between the two attributes.



(A) Balanced Dataset



(B) Imalanced Dataset

FIGURE 6.4. Mutual Information between the Public and the Private Labels

Fig. 6.4 shows the MI between all pairs of labels corresponding to public attributes listed in Table 6.1 and private attributes listed in Table 6.2. It can be seen that some pairs of attributes typically have higher MI, which means that those pairs of attributes are more correlated, either positively or negatively, than other pairs. For example, 'smiling' is highly correlated

to the private attribute ‘high cheekbones’. Likewise, ‘young’ is highly correlated to ‘bald’, ‘chubby’, and ‘double chin’. Whereas there is no correlation between the attribute pairs such as ‘smiling’ and ‘bald’. This coheres to the cognition of images in real-life scenarios.

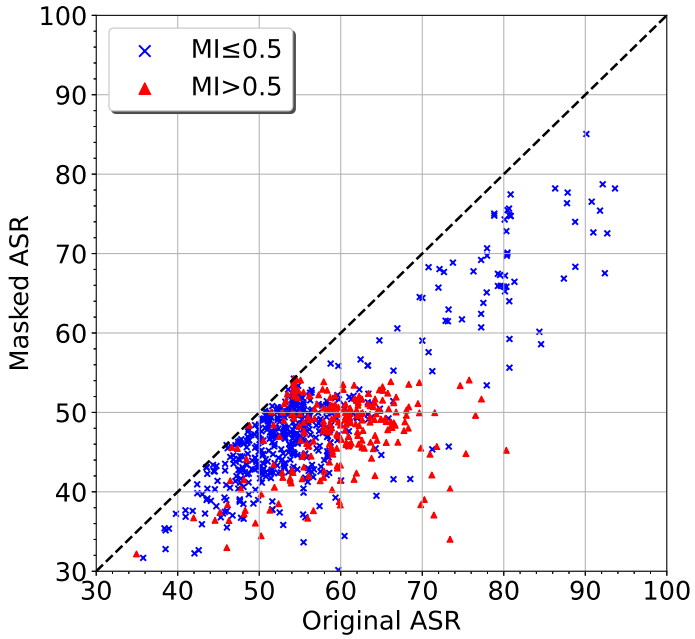
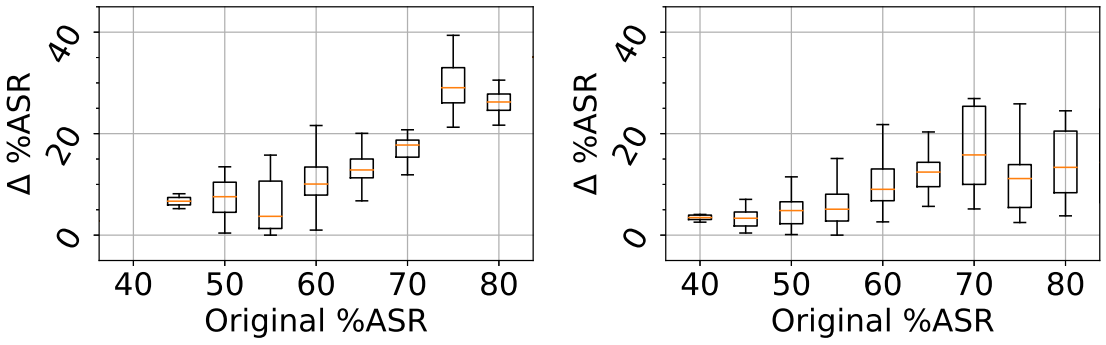


FIGURE 6.5. Original and Masked ASR for Private and Public Attributes Grouped based on Mutual Information



(A) $MI > 0.5$ (B) $MI \leq 0.5$

FIGURE 6.6. Change in ASR Against the Original ASRs for Private and Public Attributes Grouped based on Mutual Information

It is assumed that the clients have already decided on the optimal mask determination and application methods, which will be discussed in the following Sections. Fig. 6.5 shows the ASRs for the original unmasked model and the masked model, separated by MI values of 0.5.

The diagonal dash line is the boundary when the masked ASR is equal to the original ASR. The points under the diagonal line are the cases where masking reduces the ASR to a value lower than the original unmasked ASR. While all points are below the diagonal line, it can be observed that points belonging to the $MI > 0.5$ group have a more significant difference between masked and original ASRs (i.e., further away from the diagonal line).

We can see that the majority of the points with an original ASR of less than 55% belong to the $MI \leq 0.5$ group. This indicates that when the MI is lower, the PIA is less likely to succeed even without masking. However, all the cases where the masked ASR is greater than 55% also belong to the $MI \leq 0.5$ group. While their masked ASR has also decreased, the reduction is not as significant as in the $MI > 0.5$ group.

As shown in Fig. 6.6, where the $MI > 0.5$ cases in Fig. 6.6a have a more significant increase in the average ΔASR with the increasing original ASR values, compared to the $MI \leq 0.5$ cases in Fig. 6.6b. This suggests that if the private labels have a low MI with the public labels, the proposed *FlexMask* technique is likely to be less effective in protecting them. Nevertheless, many of those private labels may anyway have a lower ASR close to a random guess, making them difficult to attack. In summary, a client can estimate the effectiveness of masking for protecting their private attributes by computing the MI between the public attribute and the private attribute they wish to protect.

6.5.2 Splitting and Masking Techniques

For the splitting and masking techniques, the results are presented from the variants listed in Sec. 6.3.1 and Sec. 6.4.3. The five questions will be addressed one at a time. When addressing one question, the optimal combination for the other parameters will be selected. For each pair of the private and public attributes, The results with the largest difference between the original ASR and masked ASR, separated by the independent variables, will be selected.

Sensitivity measurements of the mask layer neurons (RQ1): As mentioned in Sec 6.4.3, various sensitivity measurements are considered when deciding which neurons to mask. Table 6.3 presents the mean and standard deviation of the ASR and VA for different sensitivity

measurement methods. We can see that the masked ASR is similar for centroid, correlation, KLD, DBS, and CHS-based sensitivity measurements. Comparatively, random masking yields the highest masked ASR, which is approximately 3% higher than the other methods. This means that the attack-defencing performance for random masking is generally the worst among all methods. Centroid, KLD, DBS, and CHS-based sensitivity measurements are similar in the masked VA. Comparatively, the correlation-based mask determination method yields the lowest masked VA (approx. 7% difference), followed by random masking (approx. 2% difference). Therefore, among the methods considered, clients should avoid correlation and random-based sensitivity measurements, as these tend to have a more significant negative impact on model utility.

TABLE 6.3. Comparison of Mean (μ) and Standard Deviation (σ) % ASR and % VA for Different Sensitivity Measurement Methods

	ASR_{masked}		ΔASR		VA_{masked}	
	μ	σ	μ	σ	μ	σ
Cen	43.33	11.01	21.08	17.67	77.06	8.43
Cor	43.12	11.21	21.39	17.91	69.61	13.87
KLD	43.69	11.63	20.50	18.37	76.48	9.04
CHS	42.95	10.85	21.00	17.63	78.69	5.48
DBS	42.92	10.66	21.20	17.49	77.93	6.93
Rand	45.94	11.75	18.07	17.27	76.01	8.58

Table 6.4 shows the computation time overheads in seconds for mask generation and training. We can see that the mask generation time for the correlation-based sensitivity measurement is lower than that of other methods except for random masking. The DBS-based method has the highest computation time for mask generation. However, as concluded above, the correlation-based method should not be considered due to its high negative impact on the model utility. Therefore, centroid and CHS-based methods are preferable as they provide a good balance between all three criteria considered (i.e., lower masked ASR, higher masked VA, and lower computation time).

Masking ratios (RQ2): We consider masking 5%, 15%, 25%, 50%, and 75% of the neurons in the chosen mask layer. Table 6.5 lists the comparison of the mean and standard deviation of the ASR and VA for different masking ratios. It shows that when the masking ratio increases

TABLE 6.4. Comparison of Average Computation Time Overheads (Second) for Different Masking Techniques

	Cen	Cor	KLD	DBS	CHS	Rand
Mask Gen	0.299	0.195	0.485	0.593	0.396	0.00004
Training	0.042	0.043	0.042	0.042	0.042	0.048

from 5% to 25%, the average ASR and VA reductions are 8.83% and 1.57%, respectively. If the masking ratio is further increased to 75%, the ASR and VA further reduce by 0.9% and 11.19%, respectively.

TABLE 6.5. Comparison of Mean (μ) and Standard Deviation (σ) % ASR and % VA between Different Masking Ratios

	ASR_{masked}		ΔASR		VA_{masked}	
	μ	σ	μ	σ	μ	σ
5%	53.37	7.66	6.53	3.40	78.53	6.16
15%	51.03	8.39	9.38	4.29	78.25	6.41
25%	44.54	12.01	19.00	18.42	76.96	7.81
50%	43.38	10.99	21.33	17.67	71.45	13.02
75%	43.64	10.89	21.66	17.43	67.34	14.45

The computation times for mask generation and training are listed in Table 6.6. The results show that if the masking ratio is increased from 5% to 75%, the training time is increased by 7.14%. The change in mask generation time for different masking ratios is negligible due to the top-k masking needs to traverse through the entire mask layer for all different masking ratios, as shown in Lines 6-10, 13-19 in Algorithm 8.

TABLE 6.6. Comparison of Average Computation Time (Second) for Different Masking Ratios

	5%	15%	25%	50%	75%
Mask Gen	0.326	0.326	0.329	0.330	0.329
Training	0.042	0.042	0.043	0.044	0.045

Therefore, among the five ratios selected, a 25% masking ratio is considered to give a good balance in reducing ASR without a significant decrease in VA.

Mask application technique (RQ3): The two mask application methods in Sec. 6.4.3 are investigated. As shown in Table 6.7, replacing the activation outputs of the masked neurons

with zeros slightly increases the mean and reduces the standard deviation of VA due to the fact that no additional random noise is added to the masked activation outputs. The results also show that replacing activation outputs with random numbers slightly increases the mean and standard deviation of the masked ASR, indicating the increase in the difficulty of protecting clients' private attributes. However, the differences in both ASR and VA are insignificant for different mask application methods.

TABLE 6.7. Comparison of Mean (μ) and Standard Deviation (σ) % ASR and % VA between Replacing the Masking Neurons with Random Numbers and Zeros

	Top-k Zero		Top-k Random	
	μ	σ	μ	σ
ASR_{masked}	42.62	11.19	42.87	11.30
ΔASR	19.20	11.11	21.34	18.46
VA_{masked}	74.83	10.93	74.18	11.25

TABLE 6.8. Comparison of Average Computation Time (Second) between Masking with Random Numbers and Zeros

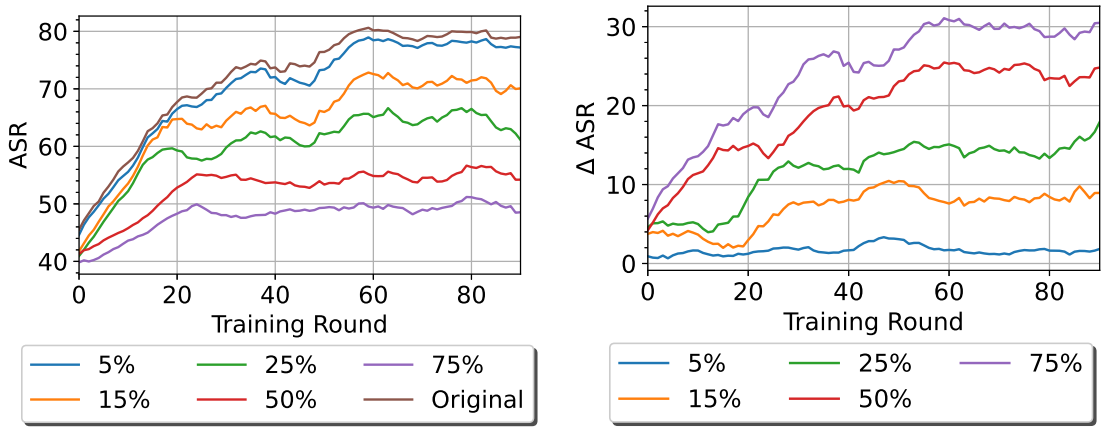
	Top-k Random	Top-k Zeros
Mask Gen	0.330	0.330
Training	0.046	0.040

As shown in Table 6.8, due to the model training incurs random number generation, masking with zeros reduces the training time overhead by approximately 13% compared to masking with random numbers. It is noted that zero masking can be applied if the clients have strict resource constraints or latency requirements.

While using zero masking increases the VA by less than 1% on average, it comes at the cost of leaking the private attribute to the adversary. If the mask layer is at the split layer, the adversary will easily reveal the pattern of the masked neurons, such that it can use the masking pattern to determine the attributes the clients intend to hide and their privacy levels. The privacy level of each client can also be estimated by counting the number of consecutive zeros presented in the activation outputs.

Mask application round (RQ4): As mentioned in Sec 6.4.3, the worst-case scenario is considered where the client only applies the mask once in a particular training round. To

explore the effect on the attack performance and masking effectiveness, the scenarios where attacking and masking are at the same training round are considered. The model is trained for different numbers of rounds before performing PIA and applying the mask. As mentioned in Sec.6.4.2, the model converges at approximately 100^{th} training round. Therefore, the training round when applying the mask ranges between one and 100. 'smiling' is selected as the public attribute and 'high cheekbones' is selected as the private attribute. The MI between public and private attributes is 0.69, indicating that the public attribute is correlated with the private attribute. The first fully-connected layer after the ReLu activation function is selected as both split and mask layers.

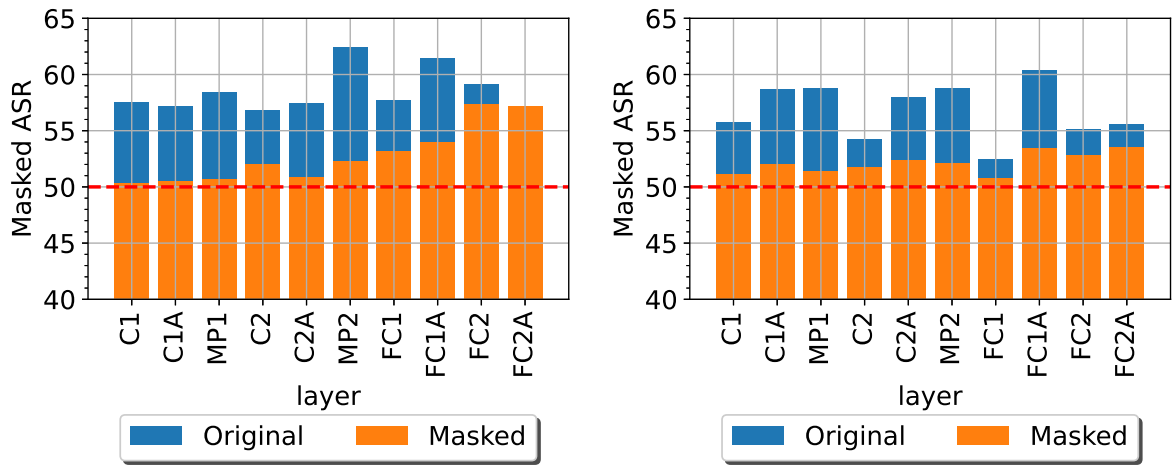


(A) ASR (B) ΔASR

FIGURE 6.7. Comparison of Masking at Different Training Rounds

As shown in Fig. 6.7a, the original ASR increases as the number of training rounds increases and becomes stable after the 60^{th} training round. The masked ASR also increases as the number of training rounds increases. Alternatively, Fig. 6.7b shows that the ΔASR increases as the number of training rounds increases. As the masking ratio increases, the masked ASR typically converges to a lower value. The results show that a well-trained model is more vulnerable to PIA due to the fact that more information about the training dataset is embedded in the trained model compared to an untrained model. The graph also shows that *FlexMask* protects the model against the PIA by reducing the ASR to a lower value compared to the original model. The average Δ ASR also increases because of the reduction in the difficulty of determining the mask when a model is well-trained.

Mask application layers (RQ5): The effect of PIA performance by the split layer selection and the effect of the attack-defencing performance by the mask layer selection are investigated. As shown in Sec.6.5.1, the MI between the private and public labels affects the original and masked ASR. Based on the previous results, centroid- and DBS-based sensitivity measurements are used to select masked neurons with 25% of the most sensitive neurons masked with random numbers. The following two ranges of MI values are considered in Fig.6.8.



(A) $MI > 0.5$ (B) $MI \leq 0.5$

FIGURE 6.8. Original and Masked ASR with Different Mutual Information

Fig. 6.8a shows the scenario with $MI > 0.5$. It can be seen that the masked ASR increases for the mask layers closer to the outputs. The lowest masked ASR is approximately 50% for the first layer after the inputs. And the masked ASR increases more significantly after the second convolutional layer. This means that masking protects the private attributes more effectively for layers closer to the inputs, especially for those layers within the feature extraction module (i.e., layers before the first fully-connected layer.

Fig. 6.8b shows the scenario with $MI \leq 0.5$. The lowest original and masked ASRs are at the first fully-connected layer, which is close to 50%. The change in the masked ASR is insignificant for different mask layers.

Table 6.9 shows the average computation overheads for mask generation and training with mask for mask layers with different numbers of neuron parameters. The number of neurons

TABLE 6.9. Comparison of Average Computation Time Overheads (Second) for Different Mask Layers

Mask Layer (Neurons)	C1 (3456)	MP1 (864)	C2 (1024)	MP2 (256)	FC1 (120)	FC2 (84)
Mask Gen	0.71	0.29	0.31	0.19	0.24	0.23
Training	0.032	0.023	0.024	0.022	0.021	0.020

in the layer is shown in the second row of the table (in brackets). Generally, as the number of parameters in the mask layer increases, the computation time for both mask generation and model training increases.

In summary, the clients can select their preferred mask layer considering the computation time and privacy preservation performance estimated by the known information of MI. When MI is lower than 0.5, the masked ASR is similar across different mask layers. Considering that the mask generation and training time overheads are also lower for the layers closer to the output layer, it is recommended to mask as close to the output layer as possible. Due to the mask layer must be within the client-side model, a client should mask at the split layer When MI is lower than 0.5 to reduce the computing time overheads and masked ASR.

Comparatively, when MI is higher than 0.5, the client can choose to mask a layer within the feature extraction module to reduce the ASR to close to 50%. Given that the mask generation and training time overheads reduce for layers closer to the outputs, the following two cases are considered:

(1) If the split layer is within the feature extraction module (before the first fully-connected layer), the mask should be applied to the split layer.
(2) If the split layer is within the classification module (after and including the first fully-connected layer), the mask should be applied to the last layer of the feature extraction module (i.e., the second max-pooling layer).

6.5.3 Dataset Distribution

To determine the impact of dataset distribution on protecting privacy, balanced and imbalanced datasets are simulated. The balanced datasets are created by selecting an equal number of celebrities holding samples with a majority of positive and negative private labels. Comparatively, the celebrities are randomly selected from the entire dataset with the distribution shown in Table 6.2 to create imbalanced datasets. Table 6.10 shows that the original ASR for the balanced dataset is similar to that for the imbalanced dataset. Alternatively, we can see that the VA for the balanced dataset is slightly higher than that for the imbalanced dataset, and the ΔVA for both balanced and imbalanced datasets is similar. This means that the vulnerability of the model to PIA is similar for different distributions of the clients' private attributes. It also shows that the average ΔASR is larger for the imbalanced dataset, indicating that masking can protect the private attribute more effectively when the private labels of the adversary's dataset are imbalanced. For both balanced and imbalanced cases, *FlexMask* reduces the average ASR to less than 50%.

TABLE 6.10. Comparison of Mean (μ) and Standard Deviation (σ) ASR and VA for Balanced and Imbalanced Datasets

		Balanced		Imbalanced	
		μ	σ	μ	σ
ASR	$ASR_{original}$	58.11	10.64	58.57	10.43
	ASR_{masked}	49.28	8.35	45.29	10.52
	ΔASR	8.83	6.47	13.27	15.39
VA	$VA_{original}$	79.37	5.16	78.52	5.57
	VA_{masked}	74.93	11.12	74.08	11.05
	ΔVA	4.44	10.61	4.44	10.30

6.5.4 Discussion

In the simulations, it is observed that *FlexMask* reduces the ASR for all the cases considered. Specifically, for cases where MI between the public and private attributes is higher than 0.5, *FlexMask* effectively reduces the ASR to less than 55%, which is close to a random guess. Among the five sensitivity measurements selected, centroid- and DBS-based measurements provide a good balance between privacy preservation, model utility and computation time

overheads. Both the ASR and VA decrease as the increase in the portion of the masked neurons. In particular, the 25% masking ratio reduces the ASR by 19% with only a 1.57% reduction in the model utility in terms of VA, compared to the conventional SL without masking. While replacing masked neurons with zero and random numbers yields similar ASR and VA, replacing them with random numbers provides better privacy due to the increase in the difficulty of revealing the masking patterns. We also observe that both the original ASR and ΔASR at an early training round are low, indicating higher attack difficulty and lower effectiveness of privacy protection. For private attributes with MI greater than 0.5, masking at layers within the classification module is less effective, whereas masking a layer closer to the output layer is preferable for private attributes with MI less than or equal to 0.5.

The simulations provide insights into the attack-defencing performance of various client-side setups of the *FlexMask* technique from an adversary's point of view and help clients decide the configurations they should use without knowing the adversary's attack model. While this chapter focused on the CelebA dataset with the LeNet-5 model as an example, the approach of deriving the optimal configuration is generalisable for different datasets and models. The decisions can be further optimised by applying a grid search for all possible combinations of the options in each question. However, this will be an NP-hard optimisation problem requiring a large amount of computing power for the simulation.

6.6 Summary

This chapter proposed *FlexMask*, a personalised masking technique to protect SL models against PIA. We considered mask determination techniques using various statistical measurements and lightweight mask application methods by replacing different ratios of activation outputs in a layer of a CNN with constant or random numbers following the original distribution. Simulation-based analyses were performed using a multi-attribute face recognition dataset to show the attack-defencing performance when masking at different training rounds and at different layers. The limitations of *FlexMask* were identified when the difficulty for masking or the PIA increases as the MI between the private and public attributes decreases.

CHAPTER 7

Personalised Distributed Transfer Learning for Resource-Constrained Scenarios

---

7.1 Introduction

In Chapter 6, the *FlexMask* technique was proposed to mask the activation outputs from neurons which are sensitive to clients' private hidden attributes. The *FlexMask* technique was based on the theory that different neurons in an ANN model are responsible for different semantic labels [199]. Previous studies also show that the information embedded in different layers of an ANN model varies [119], [207]. The early exit technique suggests that it is possible to connect an SLP classifier at an early layer of an ANN to perform various classification tasks [164], [165], [208].

Inspired by *FlexMask* and the early exit technique, this chapter develops a neuron selection technique named *FlexNS*. Instead of removing neurons in the split layer to protect the sensitive attributes, *FlexNS* selects the neurons in a layer and uses them to perform personalised training tasks. To further improve the efficiency, *FlexNS* exits the neural network early at the selected layer to reduce the number of neurons and layers traversed during the inference process.

7.2 The Proposed *FlexMask* Technique

This Section presents the proposed *FlexNS* framework based on a generic ANN structure where many neurons are connected in a layered structure. During the forward propagation process, the activation outputs from the neurons in a layer are fed into the neurons in the

adjacent layer closer to the outputs sequentially. After the forward propagation, the outputs from the last layer are compared with the truth labels to compute the results of the loss function, followed by the backward propagation of the gradients sequentially from a layer to the adjacent layer closer to the inputs to update the neurons' parameters. As illustrated in Fig. 7.1 where a LeNet-5 CNN model is used as an example, the selected neurons in the target layer (conv1) are connected to different *FlexNS* lightweight target models for classifying user-defined features for different IoT client devices.

A centrally-trained ANN model in *FlexNS* is separated into two parts at a chosen layer. This separation layer is defined as the *target layer*, and all the layers before the target layer (i.e., layers closer to the input layer than the target layer) are used as the *source model*. A subset of selected neurons in the target layer are connected to a fully-connected SLP classification module (i.e., *target model*) trained by each client on their tasks (i.e., *target tasks*). The label chosen for the target task is defined as the *target label*.

For simplicity, the neurons in one target layer and a single binary target label are selected at a time. The pre-training of the source model follows the typical ANN training process, which includes sequential forward propagation and backward propagation. Note that during the training of the *FlexNS* target model, the backward propagation only traverses through the target models, whereas no further training is required after the pre-training of the source model.

To select the neurons in the target layer, we first need to analyse the sensitivity of those neurons with respect to the target label. Then, the neurons with higher sensitivity are selected followed by the training process. The *FlexNS* process is divided into three steps: activation output separation, separation measurement calculation, and model training with neuron selection. They will be discussed separately in the following subsections.

7.2.1 Separation of Activation Outputs

The first step of computing the separation measurements is to separate the activation outputs from each neuron in the target layer with respect to the target label. Let d be the data sample

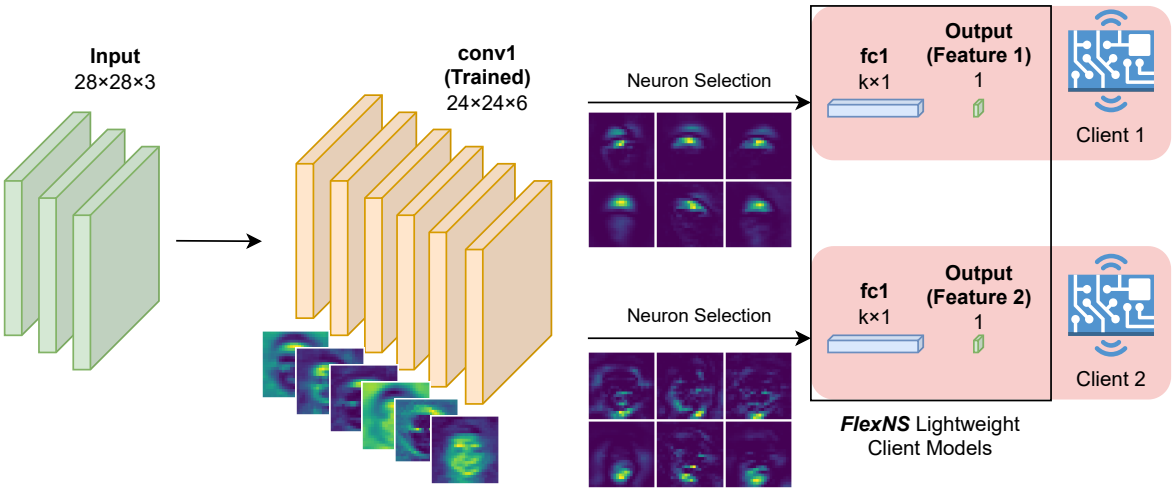


FIGURE 7.1. FlexNS Neuron Selection from the First Convolutional Layer in a LeNet-5 CNN Model

index of a dataset of size D (i.e., $d \in [1, D]$). For each training round, the activation outputs from all neurons in the target layer are denoted as:

$$\mathbf{c} = \{\mathbf{c}_1, \ldots, \mathbf{c}_d, \ldots, \mathbf{c}_D\}. \tag{7.1}$$

For a data sample d, the activation outputs $\mathbf{c}_d$ from neurons in the target layer are denoted as:

$$\mathbf{c}_d = \{e_{d,1}, ..., e_{d,r}, ..., e_{d,R}\}, \tag{7.2}$$

where r is the neuron index and R is the total number of neurons in the target layer. The target label for data sample index d is denoted as:

$$\mathbf{y} = \{y_1, ..., y_d, ..., y_D\}. \tag{7.3}$$

It is assumed a binary target label: $y_d \in \{0, 1\}\ \forall\ d \in [1, D]$, where 0 represents a negative sample and 1 represents a positive sample. The client or server separates its target layer activation outputs based on the target label. The function for separating the activation outputs is defined as $\texttt{ActSep}(\mathbf{c}, \mathbf{y})$. The function returns two sets of activation outputs belonging to positive and negative target labels.

After a training round, the activation outputs from all samples used in the training rounds are separated into two sets based on the target label using Algorithm 10:

Algorithm 10 Function for Separating Target Layer Activation Outputs According to the Target Label

Input:

- Activation outputs from target layer: $\mathbf{c} = \{\mathbf{c}_1, ..., \mathbf{c}_d, \ldots, \mathbf{c}_D\}$
- Target labels of the samples: $\mathbf{y} = \{y_1, ..., y_d, ..., y_D\}$

Output:

- Target layer activation outputs corresponding to the positive class of the target label: **c_pos**
- Target layer activation outputs corresponding to the negative class of the target label: **c_neg**

function ACTSEP($\mathbf{c}, \mathbf{y}$)
  c_pos $\leftarrow \{\}$
  c_neg $\leftarrow \{\}$
  for $d\ in\ 1, 2, ..., D$ **do** ▷ loop through the data samples
    if $y_d = 1$ **then** ▷ insert target layer outputs to the corresponding set
      c_pos.`insert`($\mathbf{c}_d$)
    else
      c_neg.`insert`($\mathbf{c}_d$)
    end if
  end for
  return c_pos, **c_neg**
end function

$$\mathbf{c\_pos} = \{\mathbf{c\_pos}_1, ..., \mathbf{c\_pos}_b, ..., \mathbf{c\_pos}_B\} \tag{7.4}$$

$$\mathbf{c\_neg} = \{\mathbf{c\_neg}_1, ..., \mathbf{c\_neg}_b, ..., \mathbf{c\_neg}_B\} \tag{7.5}$$

Note that the target label could be one of the labels of the actual classification tasks of the neural network, or a known property/feature of the input data.

7.2.2 Separation Measurement Metrics

After training the source model and separating the activation outputs from all neurons in the target layer, the next step is to rank these neurons based on their importance for the target task, or the sensitivity of the neurons with respect to the target label. Different from the encoder-decoder approach in [209], [210], Statistical-based measurements are considered to

estimate the neurons' sensitivity with respect to the target label without the need to perform computational-intensive training processes. To measure the sensitivity of each neuron in the target layer, we only look at the activation outputs from the same neuron each time. The neuron's activation outputs corresponding to samples with positive and negative target labels are separated to form two clusters in a 1-dimensional space.

Algorithm 11 Function for Computing the Separation Measures according to the Target Label

Input:

- Target layer activation outputs correspond to the positive class of the target label: **c_pos**
- Target layer activation outputs correspond to the negative class of the target label: **c_neg**

Output:

Separation measurements corresponding to each neuron: **meas**

```
3:  function SEPMEASURE(c_pos, c_neg)
4:      num_pos = c_pos.length()                    ▷ number of positive samples
5:      num_neg = c_neg.length()                    ▷ number of negative samples
6:      meas ← {}
7:      for r in 1, 2, ..., R do                    ▷ loop through neurons
8:          c_pos_r ← {c_pos_{d,r}, d ∈ [1, num_pos]}
9:          c_neg_r ← {c_neg_{d,r}, d ∈ [1, num_neg]}
10:         sep ← Sep(c_pos_r, c_neg_r)
11:         meas.insert(sep)
12:     end for
13:     return meas
14: end function
```

Algorithm 11 returns the list of separation measurements for all neurons. `Sep` function in line 10 returns the separation measurements of the activation outputs belonging to positive and negative target classes from an individual neuron. As examples, the following two commonly-used metrics are considered to measure the separation of the two clusters of samples, C_{pos} and C_{neg}.

CHS: The CHS [204] (also known as the Variance Ratio Criterion) measures the dispersion. A higher CHS value indicates better separation between clusters and denser samples within the clusters. The calculation of CHS is as follows:

The centroid of $C_{pos} \cup C_{neg}$ is defined as c_{pn} and the centroids of clusters C_{pos} and C_{neg} are c_{pos} and c_{neg}, respectively. The between-cluster dispersion is defined as:

$$\begin{aligned} B = |C_{pos}| \times \left(c_{pos} - c_{pn}\right) \times \left(c_{pos} - c_{pn}\right)^T \\ + |C_{neg}| \times \left(c_{neg} - c_{pn}\right) \times \left(c_{neg} - c_{pn}\right)^T \end{aligned} \tag{7.6}$$

The between-cluster dispersion is defined as:

$$\begin{aligned} W = \sum_{s \in C_{pos}} \left(s - c_{pos}\right) \times \left(s - c_{pos}\right)^T + \\ \sum_{s \in C_{neg}} \left(s - c_{neg}\right) \times \left(s - c_{neg}\right)^T \end{aligned} \tag{7.7}$$

Then the CHS is calculated as follows:

$$CHS = \frac{Tr(B)}{Tr(W)} \times \left(|C_{pos}| + |C_{neg}|\right) \tag{7.8}$$

DBS: DBS [205] measures the similarity between clusters. It is a distance-based metric. A lower DBS value typically indicates better separation between clusters. For cluster C_{pos} and C_{neg}, the average distance between each sample within the cluster and the centroid of the cluster is calculated as s_{pos} and s_{neg}. The distance between cluster centroids is calculated as s_{pn}. The DBS between clusters C_{pos} and C_{neg} is calculated as:

$$DBS = \frac{s_{pos} + s_{neg}}{s_{pn}} \tag{7.9}$$

7.2.3 Training with Neuron Selection

Because the server's public dataset might not include the target labels in the clients' private dataset, the separation measurements can be computed by a client or the server. A client can request the source model parameters from the server to calculate the separation measurements using its private dataset, or request the server to calculate the separation measurements using the public dataset. The server returns all the requested source model parameters to the client for training its target model. Note that unlike the conventional FL and SL, *FlexNS* only

requires the source model parameters to be transferred to a client once when the server finishes training.

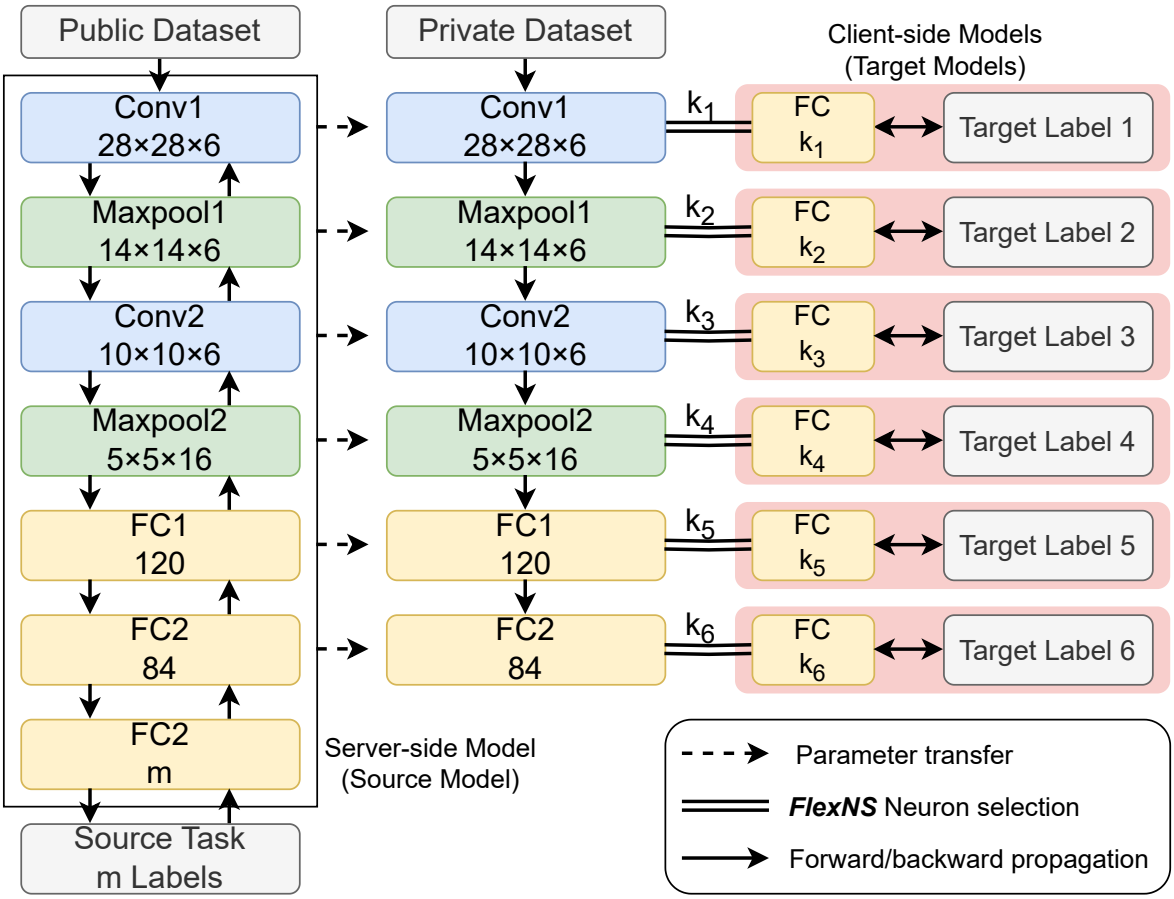


FIGURE 7.2. Overview of the *FlexNS* Training Process on a LeNet-5 CNN Model

Fig. 7.2 illustrates an example of the LeNet-5 CNN structure with *FlexNS* training process that contains:

(1) Server-side pre-training of the source model using the public dataset;

(2) Transferring source model parameters before and including the target layer to clients; and

(3) Client-side training of the target model for the target label using the private dataset with target labels. The target model contains one fully-connected layer with k neurons. Note that different clients' target models can be connected to different target layers based on their requirements.

Algorithm 12 defines the *FlexNS* training process as the collaboration between a client and the server. `ServerModelTraining` and `ClientModelTraining` are functions to train the source and target models which contain multiple forward and backward passes of the model whereas the `ForwardPassToTarget` function only performs a forward pass on the source model until the target layer and returns the activation outputs from the target layer. `TopK` function returns the index of k neurons with highest separation measurements or lowest similarity measurements. **ServerModel** is the trained source model before and includes the target layer. **ClientModel** is the trained lightweight SLP target model with a single fully-connected layer.

In the case where the separation measurement metrics are not pre-computed by the centralised server (i.e., the target labels are unavailable in the public dataset), and edge devices are unable to compute the metrics, the neurons can also be selected on a random basis. With the random selection, Algorithm 12 is executed by replacing lines 5-18 with the random selection of the **topk_idx** within the range of $[1, R]$.

7.3 Theoretical Analysis

This Section presents the theoretical analysis including computational complexity and communication cost of the proposed *FlexNS* technique.

7.3.1 Computational Complexity

In a CNN, the most computationally intensive task is to train the convolutional layer. Typically, the computational complexity for training a 2-D convolutional layer with square kernel is [211]:

$$O\left(ni \times fs^2 \times os^2\right) \tag{7.10}$$

where ni is the number of inputs, fs is the filter size (i.e., length of the side of a filter), and os is the output dimensions (i.e., length of the side of the output). Let nc be the number of convolutional layers. The total computational complexity for all convolutional layers is then

Algorithm 12 Model Training with *FlexNS*

Input:

- Number of neurons selected: k
- Public dataset: $\mathbf{f}_{pu}, \mathbf{y}_{pu}$
- Private dataset: $\mathbf{f}_{pr}, \mathbf{y}_{pr}$

function MODELTRAINING($k, \mathbf{f}_{pu}, \mathbf{f}_{pr}$)
(At the server)
  ServerModel $\leftarrow$ `ServerModelTraining`($\mathbf{f}_{pu}, \mathbf{y}_{pu}$)
  `TransferToClient`(**ServerModel**)
  if Client determines **meas then**
(At the client)
    $\mathbf{c}_{pr} \leftarrow$ `ForwardPassToTarget`($\mathbf{f}_{pr}, \mathbf{y}_{pr}$)
    (**c_pos**, **c_neg**) $\leftarrow$ `ActSep`($\mathbf{c}_{pr}, \mathbf{y}_{pr}$)
    meas $\leftarrow$ `SepMeasure`(**c_pos**, **c_neg**)
    topk_idx $\leftarrow$ `TopK`(**meas**, k)
  else
(At the client)
    `TransferToServer`(k)
(At the server)
    $\mathbf{c}_{pu} \leftarrow$ `ForwardPassToTarget`($\mathbf{f}_{pu}, \mathbf{y}_{pu}$)
    (**c_pos**, **c_neg**) $\leftarrow$ `ActSep`($\mathbf{c}_{pu}, \mathbf{y}_{pu}$)
    meas $\leftarrow$ `SepMeasure`(**c_pos**, **c_neg**)
    topk_idx $\leftarrow$ `TopK`(**meas**, k)
    `TransferToClient`(**topk_idx**)
(At the client)
    $\mathbf{c}_{pr} \leftarrow$ `ForwardPassToTarget`($\mathbf{f}_{pr}, \mathbf{y}_{pr}$)
  end if
  c_sel$_{pr} \leftarrow \mathbf{c}_{pr}$[**topk_idx**] ▷ Neuron selection
  ClientModel $\leftarrow$ `ClientModelTraining`(**c_sel**$_{pr}, \mathbf{y}_{pr}$)
end function

calculated as follows [211]:

$$O\left(\sum_{l=1}^{nc} ni_l \times fs_l^2 \times os_l^2\right) \tag{7.11}$$

Comparatively, the computational complexity for training an FC layer is:

$$O\left(ni \times no\right) \tag{7.12}$$

where ni and no are the number of inputs and outputs, respectively. In *FlexNS* target model, the number of inputs equals the number of selected neurons, i.e., $ni = k$. It is assumed

that in $FlexNS$, no (i.e., the number of classes of the target model) is significantly lower than the filter size fs and the output dimensions os. Therefore, compared to conventional FL, SL and TL, replacing d convolutional layers with a fully-connected layer at the client devices significantly reduces computational complexity. Because $FlexNS$'s neuron selection algorithm reduces ni, it further reduces the computational complexity for training the fully-connected layer compared to early exit.

Assuming that the target layer is before any fully-connected layer, the overall computational complexity of $FlexNS$ is:

$$O\left(\sum_{l=1}^{nc}\left(ni_l \times fs_l^2 \times os_l^2\right) + k \times no\right) \tag{7.13}$$

where the client-side computational complexity for FlexNS target model training is:

$$O\left(k \times no\right) \tag{7.14}$$

7.3.2 Communication Cost

In a convolutional layer, the total number of parameters is (Eq. 7.10 plus os for transferring biases):

$$O\left(\left(ni \times fs^2 \times os\right) + os\right) \tag{7.15}$$

In an FC layer, the total number of parameters is (Eq. 7.12 plus ni for transferring biases):

$$O\left(ni \times no + ni\right) \tag{7.16}$$

In FL, the centralised server coordinates the client-side training. Therefore, all model parameters need to be updated periodically. Therefore, all nu updates will upload the following volume of data to the central server. Let the number of convolutional layers be nc and the

number of fully-connected layers be nf, the communication complexity for nu updates is:

$$O\left(nu \times \left(\left(\sum_{l=1}^{nc}\left(ni_l \times fs_l^2 \times os_l\right) + os_l\right) + \left(\sum_{l=1}^{nf}\left(ni_l \times (no_l + 1)\right)\right)\right)\right) \tag{7.17}$$

In SL, the client-side training relies on the centralised server. If the model is split at layer l, the communication cost in terms of data size is:

$$O\left(nu \times no_{l-1}\right) \tag{7.18}$$

where no_{l-1} is the number of activation outputs from the last layer of the server-side model.

In $FlexNS$, if a client requests k neurons from the target layer, the communication cost in terms of data size is:

$$O\left(\left(\sum_{l=1}^{nc}\left(ni_l \times fs_l^2 \times os_l + os_l\right)\right) + k \times no + k\right) \tag{7.19}$$

Assuming that the target layer is before any fully-connected layer, nc is the number of convolutional layers the client requests. Different from FL and SL, the communication process only needs to be executed once when the server has finished training the source model.

7.4 Experimental Setup

This Section presents the experimental setup, including the hardware environment, datasets, and model used. Multiple edge client and server model training setups are simulated.

7.4.1 Hardware Environment

For source model training and large-scale simulations, Graphics Processing Units (GPUs) are used to accelerate training. Two GPU servers equipped with NVIDIA RTX A2000 and A6000 are used. The size of the Random Access Memory (RAM) is 128 GB. For the computational performance simulation, a single-board computer RPi 3B+ is used to simulate a resource-constrained client device.

7.4.2 Datasets

UNSW-NB15 [212] and Large-scale CelebFaces Attributes (CelebA) [18] datasets are used for the experiments. The UNSW-NB15 dataset is created by simulating typical network activities with nine types of attacks occurring occasionally in the network. There are 47 features in the dataset and two class labels (i.e., attack/normal and attack category). The simulations consider 36 continuous numerical features and use the attack category as class labels. The packets are divided into multiple chunks with window size of 36 to form inputs with the size of 36×36. In each chunk of 36 packets, if one of them is marked as positive for a particular type of attack, the class label for this attack type is marked as positive. All the numerical values are normalised by dividing them by the maximum value within their column. 19,400 sequential inputs are used as the server's public dataset. For each client, 500 sequential inputs are used as its private dataset. 19,400 sequential inputs are used as the validation dataset.

CelebA dataset contains annotated face images from many celebrities. There are 40 binary labels per image, describing the attributes of the image. The images are divided according to the identities of the celebrities and the size of an image is $3 \times 84 \times 84$ (three channels in RGB, 84×84 pixels). Images of 100 celebrities are selected as the public dataset to train the source model. For each client, images of 10 celebrities are selected as the private dataset to train their target model. 100 celebrities are chosen as the validation dataset.

To avoid information leakage between the server and clients, none of the data samples should overlap in the public, private, and validation datasets. The datasets are divided into multiple

batches with the size of 50 samples per batch. The validation dataset is the same for all clients and the server for fair evaluations.

7.4.3 Model

A commonly-used LeNet-5 [19] CNN model with the Adam optimiser is considered. To suit the LeNet-5 model structure, the inputs are down-sampled to 28×28 using the bilinear interpolation algorithm. The source model is a multi-label classification model, where the samples are classified by multiple labels. It is split into two parts and the first convolutional layer is selected as the target layer. The source model is pre-trained for 100 epochs before sending the source model before and including the target layer to the clients to train their target model.

Each of the target models contains one fully-connected layer with k inputs and two outputs for single-label binary classification. The target model training is performed for 100 epochs and the final validation accuracy after the 100^{th} epoch is recorded as the result.

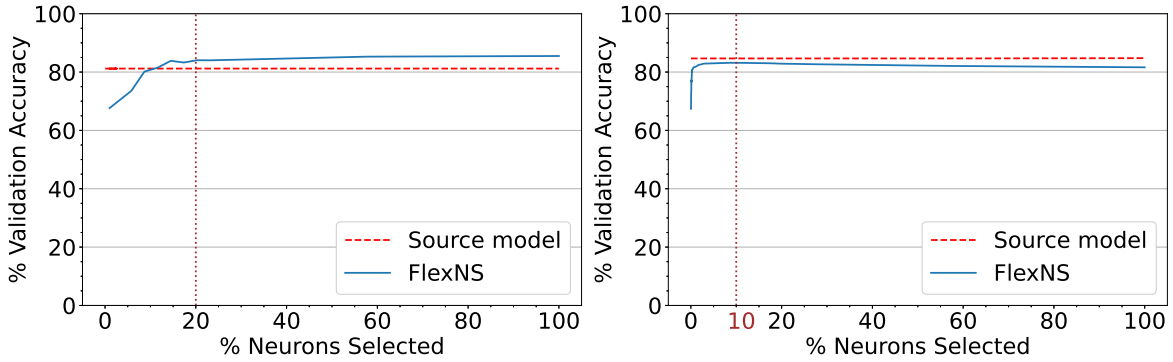
For UNSW-NB15 dataset, the learning rate is set to $1e^{-4}$ for source model training and $2e^{-4}$ for target model training. There are nine target models for classifying all nine types of attacks. For CelebA dataset, the learning rate is set to $5e^{-4}$ for both source and target model training. There are 40 target models in total for classifying all 40 binary attributes of the dataset.

7.5 Experimental Results

Using the setup described in Sec. 7.4, simulations are performed on different target tasks, measurement metrics, target layer selections, and pre-training rounds. Additionally, the model utility of the proposed *FlexNS* model is compared with the conventional structured pruning technique, and computational performance under limited computational resources with FL and SL. The simulations are repeated using five different random seeds to reduce the outliers as an effect of randomness.

7.5.1 Comparison with Source Model

To investigate the impact on model utility by replacing the source model with lightweight *FlexNS* models, CHS separation measurement described in Sec. 7.2.2 is considered to select neurons in the target layer of the source model to be connected to each target model for different classification tasks. The rest of the configuration is kept consistent except for the number of neurons selected (i.e., k). The average validation accuracies across the target models for all target labels are used as the results in Fig. 7.3.



(A) UNSW-NB15 dataset. (B) CelebA dataset.

FIGURE 7.3. Average Validation Accuracy across all Target Models in *FlexNS* for Different Target Tasks, with Different Percentages of Neurons Selected using the CHS for each Target Model

Fig. 7.3a presents the validation accuracies for different percentages of neurons selected for the UNSW-NB15 dataset. It is observed that the optimal validation accuracy of *FlexNS* is 85.5%, which is approximately 4.3% more than the source model. The validation accuracy generally increases as the percentage of neurons selected increases. The reason could be that the attack packets might be located in different areas in the inputs, which means that the increase in the number of neurons increases the probability that the attack can be detected.

It is noted that the dataset is imbalanced because it only simulates occasional attacks, and the frequency of different attack categories differs. Table 7.1 summarises the number of samples for different attack categories in the source and target datasets. Note that each sample in the public dataset could belong to zero, one, or multiple attack categories. In Table 7.1, the validation accuracy of the source and target models are separated for different attack categories

and the improvements in the validation accuracy of the target model with respect to the source model are calculated. 20% of the neurons in the target layer are selected. The table shows that for "analysis" and "DoS" attacks, the improvement in validation accuracy is more than 12%. This might be caused by the lack of samples for these two types of attacks in the public dataset, causing low validation accuracy of the source model, and training the target models for the specific target task using private datasets improves the model performance. Comparatively, transferring from a source label with fewer samples to a relatively balanced target label (e.g., "backdoors") or a source label with fewer samples to a highly imbalanced target label (e.g., "worms") would not improve the validation accuracy. Interestingly, transferring from a source label with more samples to a relatively balanced target label (e.g., "exploits") reduces the validation accuracy. But the lightweight target model still recovers 91.09% of the source model's validation accuracy in the worst case.

TABLE 7.1. Distributions of Samples Belonging to Different Attack Categories

	Samples (Source)	Samples (Target)		% Validation Accuracy		Increase in % Validation Accuracy
		Pos.	Neg.	Source	Target	
F	1845	78	422	70.73	73.88	4.45
A	211	1	499	86.15	97.06	12.66
B	226	179	321	85.46	85.59	0.15
D	705	30	470	79.75	89.47	12.19
R	1494	107	393	76.53	77.04	0.67
S	218	15	485	97.01	97.00	0.0001
W	24	3	497	99.66	99.66	0
E	2665	148	352	65.46	59.63	-8.91
G	1762	272	228	70.01	77.24	10.33

Attack categories: Fuzzers(F), Analysis(A), Backdoors(B), DoS(D), Reconnaissance(R), Shellcode(S), Worms(W), Exploits(E), Generic(G)

Fig. 7.3b presents the validation accuracies for different percentages of neurons selected for the CelebA dataset. The optimal validation accuracy for *FlexNS* is 83.17%, which is approximately 1.59% less than the source model. Interestingly, different from the UNSW-NB15 dataset, the optimal percentage of neurons selected for *FlexNS* is approximately 10%. If more than 10% of the neurons are selected, the validation accuracy reduces. This could be caused by overfitting the target models to the clients' private datasets. However, these results

are the average values across all 40 clients where different clients are interested in classifying different attributes. It should be noted that the model utility could be different for different target tasks and measurement metrics. Therefore, a further investigation of the model utilities for different attributes and measurement metrics is performed next.

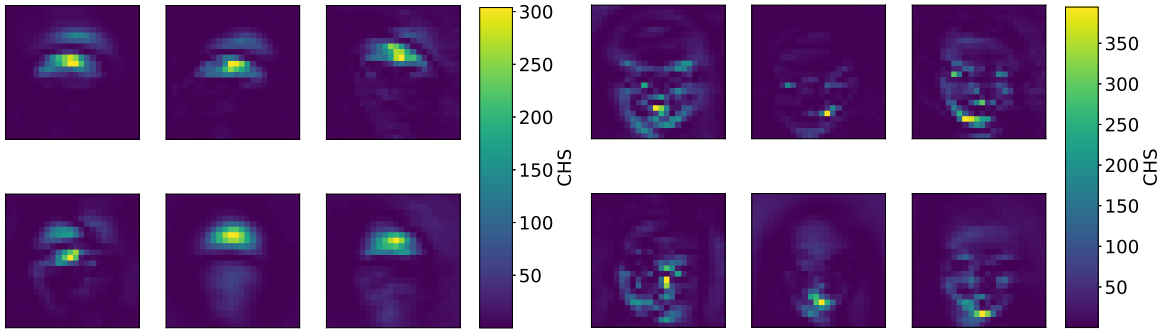
7.5.2 Attribute and Measurement Selection

In Sec. 7.5.1, the results are the average values of all 40 clients classifying different attributes. To present further insight into the results, two attributes are chosen from the CelebA dataset and the CHS is used as the measurement metric to select neurons. Fig. 7.4a shows that for attribute "bangs", the CHS at the top area of the face is higher than the rest of the areas in all channels. This means that the neurons corresponding to the top area of the face image are more likely to be selected and used for the target model for attribute "bangs". Fig. 7.5a shows that the validation accuracy for attribute "bangs" using the CHS and DBS neuron selection is higher than that for random selection. The maximum improvement in validation accuracy using deterministic selection compared to random selection is 4.53%.

Fig. 7.4b shows that for the attribute "smiling", the CHS at the eyes and mouth areas of the face is higher than the rest of the areas in all channels. This means that the neurons corresponding to the eyes and mouth areas of the face image are more likely to be selected and used for the target model for the attribute "smiling". Fig. 7.5b shows that the validation accuracy for attribute "smiling" using the CHS and CHS neuron selection is higher than that for random selection, especially for smaller percentages of neurons selected. The maximum improvement in validation accuracy using deterministic selection compared to random selection is 18.34%.

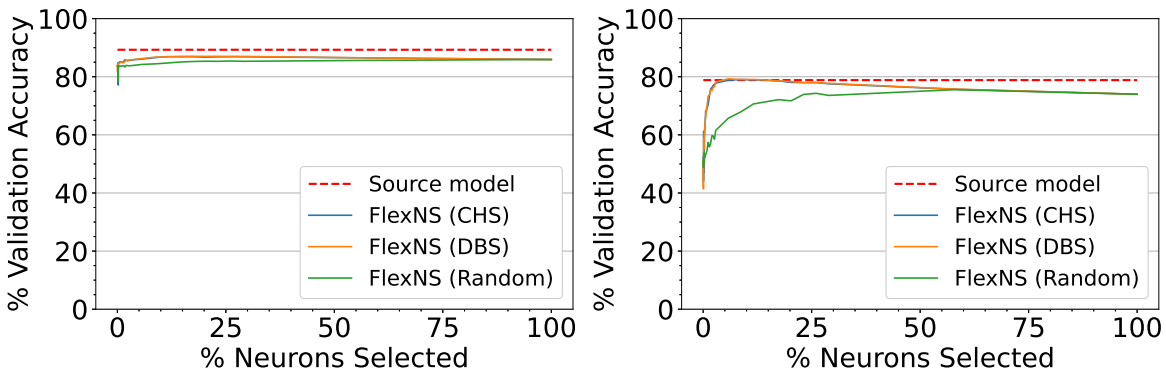
7.5.3 Target Layers

The impact on the utility of the target model by the selection of the target layer is evaluated. The model is trained using the UNSW-NB15 dataset, the CHS is used as the measurement metric for neuron selection, and neurons are selected from the same target layer at a time.



(A) "bangs" (B) "smiling"

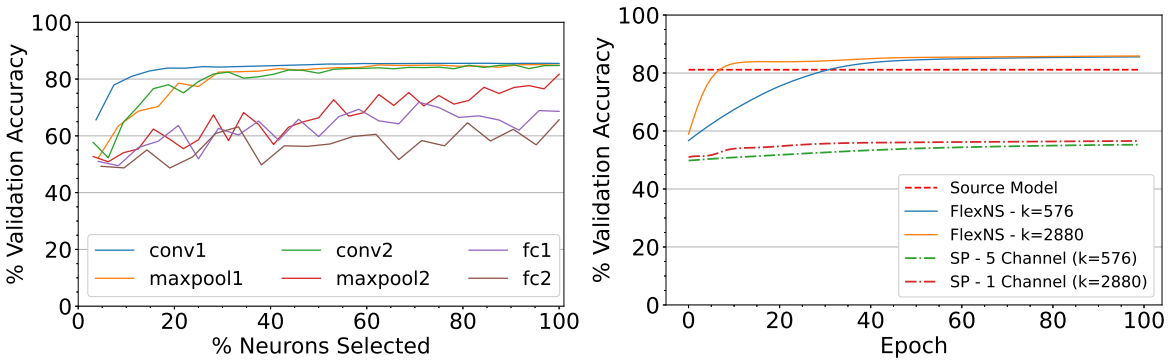
FIGURE 7.4. CHS Maps for all Six Channels in the First Convolutional Layer of the LeNet-5 CNN



(A) “bangs” (B) “smiling”

FIGURE 7.5. Single-Attribute Validation Accuracy for *FlexNS*

Fig. 7.6a shows a comparison of validation accuracy when exiting the model at different target layers. When the percentage of neurons selected is greater than 40%, selecting a target layer closer to the inputs typically yields higher validation accuracy than a layer closer to the outputs. Therefore, fewer layers need to be transferred to achieve better model utility. This is because the layers closer to the outputs are more specific to the source task, which is different from the target task. Comparatively, the layers closer to the inputs extract more general features that can be used for different classification tasks. Therefore, for devices with limited resources, selecting fewer neurons and layers is more efficient in reducing computation and communication costs. Comparatively, selecting more layers has a negative impact on the utility of the target models.



(A) Different target layers

(B) Compared with structured pruning

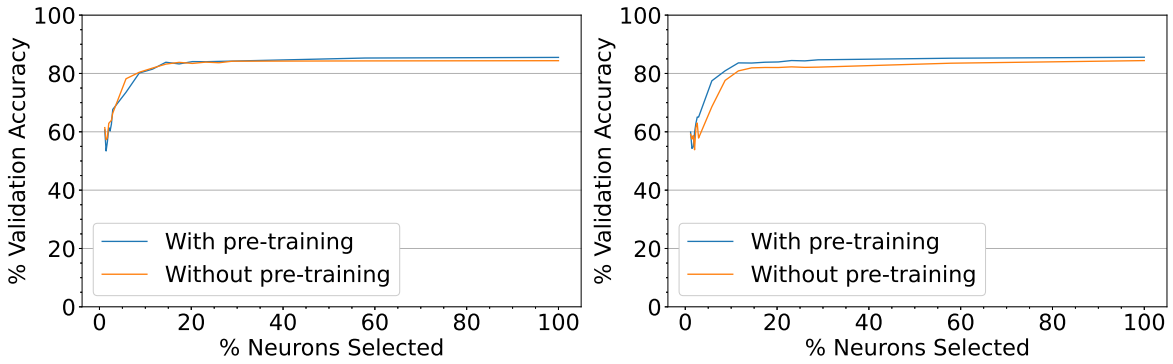
FIGURE 7.6. Validation Accuracy across all Target Models for *FlexNS*

7.5.4 Pre-training

We evaluate the performance of target models trained by leveraging a pre-trained source model or a randomly initialised source model. We use the same pre-trained source model for the UNSW-NB15 dataset as Sec. 7.5.1 and repeat the simulations five times using randomly-initialised source models generated by different random seeds. Figs. 7.7a and 7.7b show the validation accuracies for different percentages of neurons selected from a pre-trained or randomly initialised source model. We see that pre-training enhances model utility in comparison to random initialisation across most cases. For neuron selection using the CHS measurement in Fig. 7.7a, the maximum improvement is 1.27%. For random neuron selection in Fig. 7.7b, the maximum improvement reaches 8.9%. This is because the randomly-initialised model requires deterministic selection to determine the most relevant neuron to the desired attribute and the diversity of the neurons (relevant vs irrelevant) is larger than that of a pre-trained model. This aligns with the Lottery Ticket Hypothesis [213] as the deterministic selection is more likely to select neurons that contribute the most towards the desired task.

7.6 Discussion and Conclusion

This chapter proposed *FlexNS*, an edge learning framework for multi-task TL aiming to improve communication and computation efficiencies for IoT applications. In *FlexNS*, the



(A) Neuron selection using CHS. (B) Random selection.

FIGURE 7.7. Comparison between the Average Validation Accuracy across all Target Models in *FlexNS* for Using a Pre-Trained Source Model and Randomly Initialised Source Model

early exit technique is employed with flexible layer and neuron selections, ensuring that the model training can be performed on resource-constrained IoT devices with relatively good utility compared to the original model. Leveraging the model pre-trained by a powerful centralised server, the utility of the client models can be further improved. The neuron selection is flexible based on the client's privacy requirements, resource limitations and learning tasks. Experimental results showed that *FlexNS* achieves similar model utility compared to the original model for network intrusion detection and image classification tasks, with a significant reduction in training time and inference time. For different attributes, the differences in model utility improvements in *FlexNS* varied. The results also highlighted that selecting layers closer to the outputs had a negative impact on both model utility and resource consumption. Compared to the conventional structured pruning model, *FlexNS* also yielded a significantly better model utility. The *FlexNS* framework simultaneously reduced resource consumption and improved the multi-task model utility compared to the existing solutions, making it more efficient to be utilised in IoT applications.

CHAPTER 8

Conclusions and Future Work

This chapter summarises the frameworks and techniques proposed in this thesis. Limitations and potential future works are also discussed in this chapter.

8.1 Summary of Contributions

Distributed data storage and processing improves efficiency, reduces single point of failure, and preserves the privacy of the users' private information. In this thesis, five novel solutions to enhance integrity and privacy in distributed data storage and DML-based data processing frameworks were presented.

Built on top of the existing blockchain, SL and FL frameworks, the proposed approaches aim to improve trust, security and privacy while maintaining scalability. As shown in Fig. 8.1, this thesis proposed the following frameworks and techniques:

- *MapChain-D*: a scalable distributed blockchain-based data storage and communication framework;
- *VHFL*: a blockchain- and homomorphic hash-based HFL model verification technique;
- *FlexSplit*: a flexible split layer selection framework for personalised privacy preservation in SL networks
- *FlexMask*: a personalised masking-based technique for privacy preservation in SL networks; and

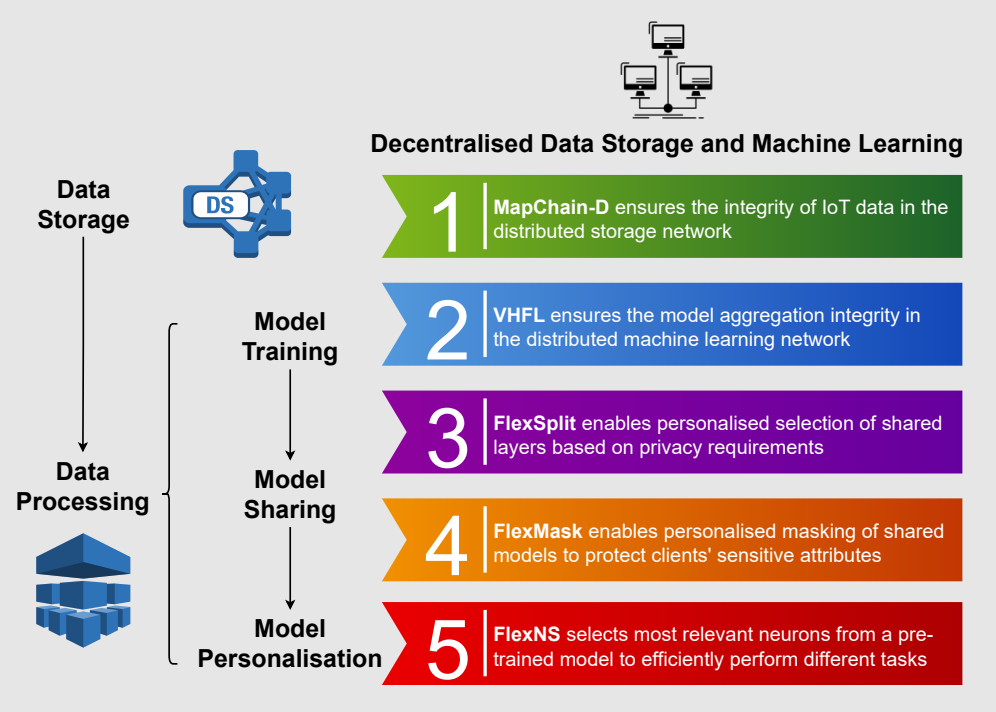


FIGURE 8.1. Proposed Frameworks and Techniques in this Thesis

- *FlexNS*: a flexible neuron selection and transfer learning technique for resource-constrained distributed machine learning.

8.2 Integration in Practical Internet-of-Things Applications

Fig. 8.1 also shows that those techniques can be adapted to multiple stages of the data flow in IoT networks. For example, after the data is collected by the IoT sensor devices, it can be stored in the *MapChain-D* network to guarantee its integrity. Then, the ML models trained by individual clients can be shared with others and aggregated hierarchically in the system using *VHFL* to guarantee the model integrity. *FlexSplit* and *FlexMask* can be integrated during model-sharing processes to preserve clients' privacy by removing or reducing sensitive information embedded in the shared model parameters. Finally, the clients can leverage the globally trained and aggregated model with the *FlexNS* algorithm to select the most relevant neurons for their individual tasks.

In practical scenarios, the techniques proposed in this thesis can be integrated into practical Industrial IoT (IIoT) applications such as smart agriculture and smart energy grid.

8.2.1 Smart Agriculture Application

Figure 8.2 shows an example of a smart agriculture application (e.g., soil quality study) using the techniques proposed in this thesis. The sensors installed on two farms collect different types of data and transmit their raw data to the base station installed in the farms. The devices in different farms aim to train a shared model with different tasks, and they wish to contribute to the global model and benefit from other clients' local training. However, the raw data is confidential and should not be circulated outside the farm where the data is collected. Therefore, the techniques proposed in this thesis can be applied to the following steps in various combinations:

- The trained model parameters are stored in the *MapChain-D* distributed storage system, and shared among different farms. The client devices at the farms retrieve the model update from the cloud server via the *MapChain-D* network.
- The cloud or edge servers perform model verification using the *VHFL* technique to ensure the integrity of the model parameters it receives. Then, they perform model aggregation hierarchically. Finally, the clients perform model verification using the *VHFL* technique to verify the correctness of the aggregated model they received.
- Before uploading the model parameters to the storage network, the farms can apply the *FlexSplit* or *FlexMask* mechanism to the shared model, depending on their model training scheme and security requirements.
- The clients can further train the model according to their desired tasks using the *FlexNS* technique.

8.2.2 Smart Energy Grid Application

Figure 8.2 shows an example of a smart grid application (e.g., distributed power generation) using the techniques proposed in this thesis. The smart meters installed on two sites collect

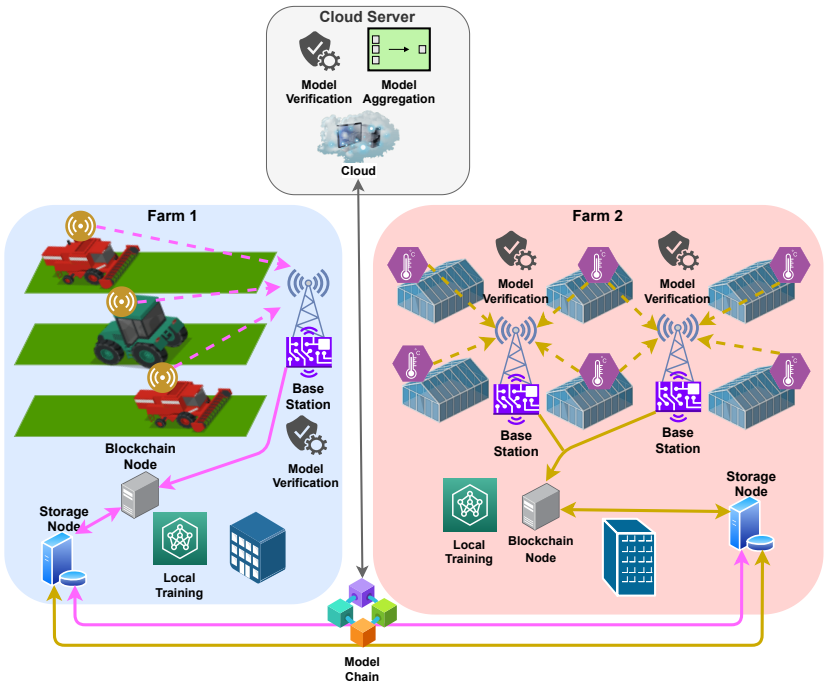


FIGURE 8.2. Example of Integration in Smart Agriculture Application

the same type of data and transfer the raw data to the base stations located within their site. The goal is to train a shared model using the data collected by all the smart meters across different sites. There are multiple devices across the sites wishing to utilise the collaboratively trained model for different tasks. The techniques proposed in this thesis can be applied to the following steps in various combinations:

- The trained model parameters are stored in the *MapChain-D* distributed storage system, allowing devices at different sites to retrieve the trained model from the *MapChain-D* network.
- The cloud or edge servers perform model verification using the *VHFL* technique to ensure the integrity of the models it receives. Then, they aggregate the models hierarchically. Finally, the clients perform model verification using the *VHFL* technique to verify the correctness of the aggregated model they received.
- Before uploading the model parameters to the storage network, the site servers can apply the *FlexSplit* or *FlexMask* mechanism to the shared model depending on their model training scheme and security requirements.

- The clients can further train the model according to their desired tasks using the *FlexNS* technique.

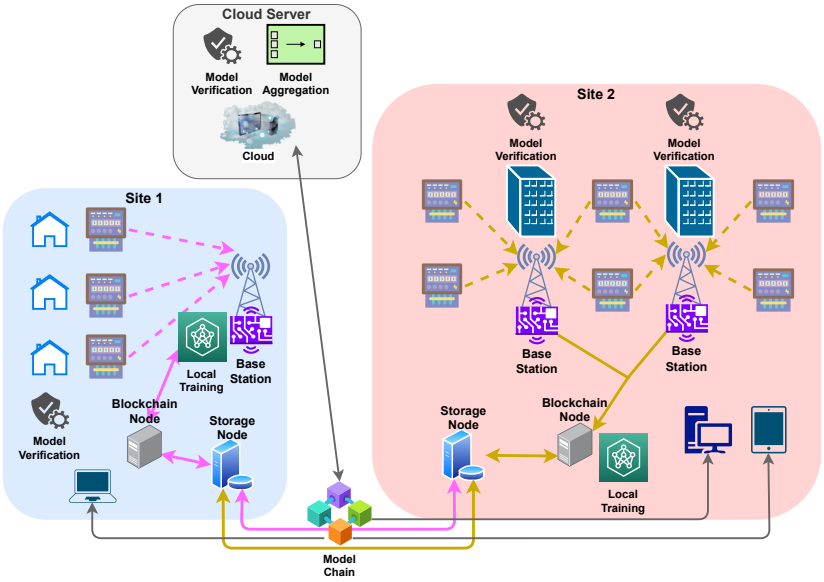


FIGURE 8.3. Example of Integration in Smart Grid Application

8.3 Limitations and Potential Future Work

8.3.1 Blockchain System

The blockchains used in *MapChain-D* were based on Ethereum to demonstrate the framework. However, Ethereum was not originally designed for data storage. The IoT sensor and actuator data might be relatively smaller in size, making the overheads negligible compared to the traditional use-case scenario of storing transactions in Ethereum blockchains. However, if we consider larger data (e.g., high-resolution images or videos), Ethereum might not be efficient for data storage. Blockchain systems dedicated to distributed file storage [214], such as Filecoin and Metadisk, can be considered to be used as the data chain.

8.3.2 Hierarchical Structure and Heterogeneity

In *VHFL* and *FlexSplit*, hierarchical structures for FL and SL were considered. However, the simulation experiments only considered a three-layer structure with only one intermediate layer of edge servers. To verify the generalisation and scalability of the proposed techniques, multiple intermediate layers of edge servers can be considered.

The efficiency and effectiveness of model verification and privacy preservation in more practical heterogeneous scenarios would be interesting to investigate further in the future. The heterogeneities in hierarchical DML systems include:

- An imbalanced hierarchical structure with different hierarchical layers for different groups with different numbers of clients.
- Statistical heterogeneity between the datasets belonging to different groups of clients. For example, the quality of data collected by groups of devices located in different areas.
- Heterogeneity in trust and security between different groups of clients and edge servers. The devices in different groups might be deployed at different network and physical security levels.
- Heterogeneities in resources, including communications and computation resources.
- Reliability such as drop-out rates and variations in communications latencies for different groups of clients and edge servers.

For instance, for groups with higher security levels and stricter resource constraints, the verification might be performed with less precision and lower frequency to reduce resource consumption.

8.3.3 Model Structure and Datasets

The model structure considered in the simulations for *VHFL*, *FlexSplit*, *FlexMask* and *FlexNS* was a conventional LeNet-5 CNN model. This was due to the common resource limitation when simulating a DML network in resource-constrained IoT scenarios. In the future, a

more advanced or complex ANN model can be adopted to verify the model robustness of the proposed frameworks. However, this future work requires additional time and resources to perform. DML for training large language models would be a potential future work with more challenges, especially for deployment in large-scale resource-constrained IoT systems.

The datasets considered in the simulations for *VHFL*, *FlexSplit*, *FlexMask* and *FlexNS* included image classification and network intrusion detection datasets. The simulations can be expanded by considering more datasets with different statistical properties to verify the robustness of the proposed framework for different datasets. For *VHFL* and *FlexSplit*, the hierarchical structure usually introduces statistical heterogeneity of the data samples from clients in different groups at different hierarchical levels. Therefore, datasets considering statistical heterogeneity can be generated to simulate a more practical scenario. In *FlexMask* and *FlexNS*, the client-side binary classification models can be extended to more generalised multi-class classification models.

8.3.4 Optimisation and Fairness Problems

FlexSplit, *FlexMask* and *FlexNS* provide clients with more flexible splitting, masking and neuron selection solutions. However, this comes at the cost of increasing complexity in finding a global optimal for all adjustable parameters, such as split layer, number of masked or selected neurons and selection criteria. The combinatoric optimisation problems are likely to be NP-hard and it is infeasible for resource-constrained client devices to test all possible combinations. The solution in this thesis was to optimise one parameter at a time. However, the order of the optimisation would strongly influence the final decision.

Moreover, as discussed in Sec. 8.3.2, the heterogeneity of clients introduces additional complexity. Additionally, providing clients with more flexibility might encourage clients to protect their privacy or save their computing resources by contributing less or even contributing negatively to the system. The clients might be selfish or ambitious, where they might wish to benefit from the global model training without contribution. Therefore, ensuring fairness

and encouraging clients' positive contributions towards the global model utility would be important for future work.

8.3.5 Real-World Deployments

While there are IoT devices such as Arduino and Raspberry Pi involved in the simulations in this thesis, the simulations are conducted in a controlled environment (e.g., good network environment, no power constraints). The practicality of the thesis can be further improved by deploying the proposed frameworks in real-world scenarios.

Specifically, the following factors can be considered for more practical deployments:

- Using data collected by actual sensors – Data collected by different sensors might have different sizes and intervals. As a result, the volume of data and latency of different sensors will differ. The impact of such heterogeneity is important to evaluate the robustness of the frameworks.
- Deployment in practical network conditions – In practice, IoT devices are deployed in heterogeneous network conditions. The latency and bandwidth differ under different network conditions. Other stability issues, such as packet loss, jittering and congestion, might also be considered in real-life implementations.
- Energy consumption – Practical IoT devices are often deployed under different power constraints. In particular, some devices are not connected to the mains electricity. Those devices are operated using battery or harvested energy (e.g., solar, kinetic, thermal), making the energy very precious to them. Therefore, optimisations on energy consumption are important to be considered in practical IoT deployments to minimise the additional energy usage.

Deploying the solution in real-world scenarios will further highlight the feasibility and effectiveness of the solutions and their robustness under realistic conditions.

8.3.6 Theoretical Analysis

The theoretical analyses for the solutions in this thesis can be further extended to include more thorough and formal analyses. Especially for the trade-off problems such as privacy, model utility and resource consumption. Optimisation criteria can be added to the optimisation problems to highlight the practicability of the solutions under different constraints.

References

[1] Z. Lyu, J. J. Park, J. Shen, H. Song and Y. Zhang, 'Guest editorial: Generative artificial intelligence at the edge in the modern internet of things,' *IEEE Internet of Things Magazine*, vol. 7, no. 3, pp. 12–14, 2024. DOI: 10.1109/MIOT.2024.10517501.

[2] L. Tawalbeh, F. Muheidat, M. Tawalbeh and M. Quwaider, 'Iot privacy and security: Challenges and solutions,' *Applied Sciences*, vol. 10, no. 12, 2020, ISSN: 2076-3417. DOI: 10.3390/app10124102. [Online]. Available: https://www.mdpi.com/2076-3417/10/12/4102.

[3] L. Jiang, L. D. Xu, H. Cai, Z. Jiang, F. Bu and B. Xu, 'An iot-oriented data storage framework in cloud computing platform,' *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1443–1451, 2014. DOI: 10.1109/TII.2014.2306384.

[4] T. Wang, J. Zhou, A. Liu, M. Z. A. Bhuiyan, G. Wang and W. Jia, 'Fog-based computing and storage offloading for data synchronization in iot,' *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4272–4282, 2019. DOI: 10.1109/JIOT.2018.2875915.

[5] J. Zou, Y. Han and S.-S. So, 'Overview of artificial neural networks,' in *Artificial Neural Networks: Methods and Applications*, D. J. Livingstone, Ed. Totowa, NJ: Humana Press, 2009, pp. 14–22, ISBN: 978-1-60327-101-1. DOI: 10.1007/978-1-60327-101-1_2. [Online]. Available: https://doi.org/10.1007/978-1-60327-101-1_2.

[6] C. Ma, J. Li, K. Wei *et al.*, '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. DOI: 10.1109/JPROC.2023.3306773.

[7] M. Le, T. Huynh-The, T. Do-Duy, T.-H. Vu, W.-J. Hwang and Q.-V. Pham, 'Applications of distributed machine learning for the internet-of-things: A comprehensive

survey,' *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2024. DOI: 10.1109/COMST.2024.3427324.

[8] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen and S. Shieh, 'Iot security: Ongoing challenges and research opportunities,' in *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, 2014, pp. 230–234. DOI: 10.1109/SOCA.2014.58.

[9] M. J. M. Chowdhury, M. S. Ferdous, K. Biswas *et al.*, 'A comparative analysis of distributed ledger technology platforms,' *IEEE Access*, vol. 7, pp. 167 930–167 943, 2019. DOI: 10.1109/ACCESS.2019.2953729.

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y. Arcas, 'Communication-Efficient Learning of Deep Networks from Decentralized Data,' in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, A. Singh and J. Zhu, Eds., ser. Proceedings of Machine Learning Research, vol. 54, PMLR, Apr. 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html.

[11] O. Gupta and R. Raskar, 'Distributed learning of deep neural network over multiple agents,' *J. Netw. and Comput. Appl.*, vol. 116, pp. 1–8, 2018, ISSN: 1084-8045.

[12] P. Vepakomma, O. Gupta, T. Swedish and R. Raskar, *Split learning for health: Distributed deep learning without sharing raw patient data*, 2018. arXiv: 1812.00564.

[13] S. J. Pan and Q. Yang, 'A survey on transfer learning,' *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010. DOI: 10.1109/TKDE.2009.191.

[14] P. Maymounkov and D. Mazières, 'Kademlia: A peer-to-peer information system based on the xor metric,' in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek and A. Rowstron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65, ISBN: 978-3-540-45748-0.

[15] P. Szilágyi, *EIP-225: Clique proof-of-authority consensus protocol*, https://eips.ethereum.org/EIPS/eip-225/, 2017.

[16] C. Gkantsidis and P. Rodriguez Rodriguez, 'Cooperative security for network coding file distribution,' in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, 2006, pp. 1–13. DOI: 10.1109/INFOCOM.2006.233.

[17] F. Mo, A. Borovykh, M. Malekzadeh, S. Demetriou, D. Gündüz and H. Haddadi, *Quantifying and localizing usable information leakage from neural network gradients*, 2022. arXiv: 2105.13929 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2105.13929.

[18] Z. Liu, P. Luo, X. Wang and X. Tang, 'Deep learning face attributes in the wild,' in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[19] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, 'Gradient-based learning applied to document recognition,' *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.

[20] T. Fernando, S. Sridharan, S. Denman and C. Fookes, 'Split 'n' merge net: A dynamic masking network for multi-task attention,' *Pattern Recognition*, vol. 126, p. 108 551, 2022, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2022.108551. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320322000322.

[21] L. Yang, J. Zhang, D. Chai *et al.*, 'Practical and secure federated recommendation with personalized mask,' in *Trustworthy Federated Learning*, R. Goebel, H. Yu, B. Faltings, L. Fan and Z. Xiong, Eds., Cham: Springer International Publishing, 2023, pp. 33–45, ISBN: 978-3-031-28996-5.

[22] I. Ergun, H. U. Sami and B. Guler, *Sparsified secure aggregation for privacy-preserving federated learning*, 2021. arXiv: 2112.12872 [cs.LG].

[23] P. Cui, U. Guin, A. Skjellum and D. Umphress, 'Blockchain in iot: Current trends, challenges, and future roadmap,' *Journal of Hardware and Systems Security*, vol. 3, pp. 338–364, 2019.

[24] S. K. Lo, Y. Liu, S. Y. Chia *et al.*, 'Analysis of blockchain solutions for iot: A systematic literature review,' *IEEE Access*, vol. 7, pp. 58 822–58 835, 2019. DOI: 10.1109/ACCESS.2019.2914675.

[25] R. Li, T. Song, B. Mei, H. Li, X. Cheng and L. Sun, 'Blockchain for large-scale internet of things data storage and protection,' *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 762–771, 2019. DOI: 10.1109/TSC.2018.2853167.

[26] G. Becker, 'Merkle signature schemes, merkle trees and their cryptanalysis,' *Ruhr-University Bochum, Tech. Rep*, vol. 12, p. 19, 2008.

[27] Z. Ullah, B. Raza, H. Shah, S. Khan and A. Waheed, 'Towards blockchain-based secure storage and trusted data sharing scheme for iot environment,' *IEEE Access*, vol. 10, pp. 36 978–36 994, 2022. DOI: 10.1109/ACCESS.2022.3164081.

[28] J. H. Khor, M. Sidorov and P. Y. Woon, 'Public blockchains for resource-constrained iot devices—a state-of-the-art survey,' *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 11 960–11 982, 2021. DOI: 10.1109/JIOT.2021.3069120.

[29] A. Reyna, C. Martín, J. Chen, E. Soler and M. Díaz, 'On blockchain and its integration with iot. challenges and opportunities,' *Future Generation Computer Systems*, vol. 88, pp. 173–190, 2018, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.05.046. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17329205.

[30] J. Xiaomeng, Z. Fan, L. Shenwen, Y. Jinglin and H. Ketai, 'Data analysis of bitcoin blockchain network nodes,' in *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2020, pp. 1891–1895. DOI: 10.1109/ICIEA48937.2020.9248092.

[31] R. Norvill, B. B. Fiz Pontiveros, R. State and A. Cullen, 'Ipfs for reduction of chain size in ethereum,' in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1121–1128. DOI: 10.1109/Cybermatics_2018.2018.00204.

[32] E. Daniel and F. Tschorsch, 'Ipfs and friends: A qualitative comparison of next generation peer-to-peer data networks,' *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 31–52, 2022. DOI: 10.1109/COMST.2022.3143147.

[33] M. F. Kaashoek and D. R. Karger, 'Koorde: A simple degree-optimal distributed hash table,' in *Peer-to-Peer Systems II*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 98–107, ISBN: 978-3-540-45172-3.

[34] P. Maymounkov and D. Mazières, 'Kademlia: A peer-to-peer information system based on the XOR metric,' in *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2429, Springer, 2002, pp. 53–65. DOI: 10.1007/3-540-45748-8_5. [Online]. Available: https://doi.org/10.1007/3-540-45748-8%5C_5.

[35] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma and S. Lim, 'A survey and comparison of peer-to-peer overlay network schemes,' *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005. DOI: 10.1109/COMST.2005.1610546.

[36] J. Aspnes and G. Shah, 'Skip graphs,' *ACM Trans. Algorithms*, vol. 3, no. 4, 37–es, Nov. 2007, ISSN: 1549-6325. DOI: 10.1145/1290672.1290674. [Online]. Available: https://doi.org/10.1145/1290672.1290674.

[37] R. Abe, S. Suzuki and J. Murai, 'Mitigating bitcoin node storage size by dht,' in *Proceedings of the 14th Asian Internet Engineering Conference*, ser. AINTEC '18, Bangkok, Thailand: Association for Computing Machinery, 2018, pp. 17–23, ISBN: 9781450361316. DOI: 10.1145/3289166.3289169. [Online]. Available: https://doi.org/10.1145/3289166.3289169.

[38] Y. Hassanzadeh-Nazarabadi, A. Küpçü and Ö. Özkasap, 'Lightchain: Scalable dht-based blockchain,' *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 10, pp. 2582–2593, Oct. 2021, ISSN: 1045-9219. DOI: 10.1109/TPDS.2021.3071176. [Online]. Available: https://doi.org/10.1109/TPDS.2021.3071176.

[39] Y. Zuo, S. Jin and S. Zhang, 'Blockchain storage, computation offloading, and user association for heterogeneous cellular networks,' *IEEE Internet of Things Journal*, vol. 9, no. 11, pp. 8191–8204, 2022. DOI: 10.1109/JIOT.2021.3113366.

[40] Y. Yu, S. Liu, P. L. Yeoh, B. Vucetic and Y. Li, 'Layerchain: A hierarchical edge-cloud blockchain for large-scale low-delay industrial internet of things applications,' *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 5077–5086, 2021. DOI: 10.1109/TII.2020.3016025.

[41] Y. Fan, T. Qiu, L. Zhang *et al.*, 'Dlbn: Group storage mechanism based on double-layer blockchain network,' *IEEE Internet of Things Journal*, vol. 9, no. 20, pp. 19 649–19 659, 2022. DOI: 10.1109/JIOT.2022.3170496.

[42] T. Xu, T. Qiu, D. Hu, C. Mu, Z. Wan and W. Liu, 'A scalable two-layer blockchain system for distributed multicloud storage in iiot,' *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 9173–9183, 2022. DOI: 10.1109/TII.2022.3179733.

[43] Z. Liao, S. Cheng, J. Zhang, W. Wu, J. Wang and P. K. Sharma, 'Gpdb: A graph-partition based storage strategy for dag-blockchain in edge-cloud iiot,' *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2022. DOI: 10.1109/TII.2022.3162201.

[44] S. Liu, J. Wu and C. Long, 'Iot meets blockchain: Parallel distributed architecture for data storage and sharing,' in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1355–1360. DOI: 10.1109/Cybermatics_2018.2018.00233.

[45] G. Zyskind, O. Nathan and A. '. Pentland, 'Decentralizing privacy: Using blockchain to protect personal data,' in *2015 IEEE Security and Privacy Workshops*, 2015, pp. 180–184. DOI: 10.1109/SPW.2015.27.

[46] H. Shafagh, L. Burkhalter, A. Hithnawi and S. Duquennoy, 'Towards blockchain-based auditable storage and sharing of iot data,' in *Proceedings of the 2017 on Cloud Computing Security Workshop*, ser. CCSW '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 45–50, ISBN: 9781450352048. DOI: 10.1145/3140649.3140656. [Online]. Available: https://doi.org/10.1145/3140649.3140656.

[47] V. K. C. Ramesh, Y. Kim and J.-Y. Jo, 'Secure iot data management in a private ethereum blockchain,' in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2020, pp. 369–375. DOI: 10.1109/COMPSAC48688.2020.0-219.

[48] 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)*, 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9148862.

[49] 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. DOI: 10.1109/TWC.2020.3003744.

[50] F. P.-C. Lin, S. Hosseinalipour, S. S. Azam, C. G. Brinton and N. Michelusi, 'Semi-decentralized federated learning with cooperative d2d local model aggregations,' *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3851–3869, 2021. DOI: 10.1109/JSAC.2021.3118344.

[51] S. R. Pandey, M. N. H. Nguyen, T. N. Dang *et al.*, 'Edge-assisted democratized learning toward federated analytics,' *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 572–588, 2022. DOI: 10.1109/JIOT.2021.3085429.

[52] F. De Rango, A. Guerrieri, P. Raimondo and G. Spezzano, 'A novel edge-based multi-layer hierarchical architecture for federated learning,' in *2021 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)*, 2021, pp. 221–225. DOI: 10.1109/DASC-PICom-CBDCom-CyberSciTech52372.2021.00047.

[53] Z. Wang, H. Xu, J. Liu, H. Huang, C. Qiao and Y. Zhao, 'Resource-efficient federated learning with hierarchical aggregation in edge computing,' in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10. DOI: 10.1109/INFOCOM42981.2021.9488756.

[54] C. Briggs, Z. Fan and P. Andras, 'Federated learning with hierarchical clustering of local updates to improve training on non-iid data,' in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–9. DOI: 10.1109/IJCNN48605.2020.9207469.

[55] 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)*, 2020, pp. 344–347. DOI: 10.1109/EuroSPW51379.2020.00053.

[56] L. Zhu, Z. Liu and S. Han, 'Deep leakage from gradients,' in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/60a6c4002cc7b29142def8871531281a-Paper.pdf.

[57] L. Shi, J. Shu, W. Zhang and Y. Liu, 'Hfl-dp: Hierarchical federated learning with differential privacy,' in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–7. DOI: 10.1109/GLOBECOM46510.2021.9685644.

[58] M. Gharibi, S. Bhagavan and P. Rao, 'Federatedtree: A secure serverless algorithm for federated learning to reduce data leakage,' in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 4078–4083. DOI: 10.1109/BigData52589.2021.9672039.

[59] M. Asad, A. Moustafa and C. Yu, 'A critical evaluation of privacy and security threats in federated learning,' *Sensors*, vol. 20, no. 24, 2020, ISSN: 1424-8220. DOI: 10.3390/s20247182. [Online]. Available: https://www.mdpi.com/1424-8220/20/24/7182.

[60] M. Fang, X. Cao, J. Jia and N. Z. Gong, 'Local model poisoning attacks to byzantine-robust federated learning,' in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20, USA: USENIX Association, 2020, ISBN: 978-1-939133-17-5.

[61] D. Cao, S. Chang, Z. Lin, G. Liu and D. Sun, 'Understanding distributed poisoning attack in federated learning,' in *2019 IEEE 25th International Conference on*

Parallel and Distributed Systems (ICPADS), 2019, pp. 233–239. DOI: 10.1109/ICPADS47876.2019.00042.

[62] J. Zhang, J. Chen, D. Wu, B. Chen and S. Yu, 'Poisoning attack in federated learning using generative adversarial nets,' in *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2019, pp. 374–380. DOI: 10.1109/TrustCom/BigDataSE.2019.00057.

[63] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin and V. Shmatikov, 'How to backdoor federated learning,' in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, S. Chiappa and R. Calandra, Eds., ser. Proceedings of Machine Learning Research, vol. 108, PMLR, Aug. 2020, pp. 2938–2948. [Online]. Available: https://proceedings.mlr.press/v108/bagdasaryan20a.html.

[64] X. Cao, M. Fang, J. Liu and N. Z. Gong, 'Fltrust: Byzantine-robust federated learning via trust bootstrapping,' in *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium*, 2021.

[65] Y. Wang and B. Kantarci, 'Reputation-enabled federated learning model aggregation in mobile platforms,' in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6. DOI: 10.1109/ICC42927.2021.9500928.

[66] W. Wan, J. Lu, S. Hu, L. Y. Zhang and X. Pei, 'Shielding federated learning: A new attack approach and its defense,' in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, 2021, pp. 1–7. DOI: 10.1109/WCNC49053.2021.9417334.

[67] J. Guo, Z. Liu, K.-Y. Lam, J. Zhao, Y. Chen and C. Xing, *Secure weighted aggregation for federated learning*, 2021. arXiv: 2010.08730 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2010.08730.

[68] X. Zhang, A. Fu, H. Wang, C. Zhou and Z. Chen, 'A privacy-preserving and verifiable federated learning scheme,' in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9148628.

[69] A. Fu, X. Zhang, N. Xiong, Y. Gao, H. Wang and J. Zhang, 'Vfl: A verifiable federated learning with privacy-preserving for big data in industrial iot,' *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3316–3326, 2022. DOI: 10.1109/TII.2020.3036166.

[70] D. Fiore, R. Gennaro and V. Pastro, 'Efficiently verifiable computation on encrypted data,' in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14, Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 844–855, ISBN: 9781450329576. DOI: 10.1145/2660267.2660366. [Online]. Available: https://doi.org/10.1145/2660267.2660366.

[71] X. Guo, Z. Liu, J. Li *et al.*, 'Verifl: Communication-efficient and fast verifiable aggregation for federated learning,' *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1736–1751, 2021. DOI: 10.1109/TIFS.2020.3043139.

[72] G. Xu, H. Li, S. Liu, K. Yang and X. Lin, 'Verifynet: Secure and verifiable federated learning,' *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2020. DOI: 10.1109/TIFS.2019.2929409.

[73] X. Zhang, F. Li, Z. Zhang, Q. Li, C. Wang and J. Wu, 'Enabling execution assurance of federated learning at untrusted participants,' in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 1877–1886. DOI: 10.1109/INFOCOM41043.2020.9155414.

[74] S. Warnat-Herresthal, H. Schultze, K. L. Shastry *et al.*, 'Swarm learning for decentralized and confidential clinical machine learning,' *Nature*, vol. 594, no. 7862, pp. 265–270, 2021.

[75] D. C. Nguyen, M. Ding, Q.-V. Pham *et al.*, 'Federated learning meets blockchain in edge computing: Opportunities and challenges,' *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12 806–12 825, 2021. DOI: 10.1109/JIOT.2021.3072611.

[76] C. Ma, J. Li, L. Shi *et al.*, 'When federated learning meets blockchain: A new distributed learning paradigm,' *IEEE Computational Intelligence Magazine*, vol. 17, no. 3, pp. 26–33, 2022. DOI: 10.1109/MCI.2022.3180932.

[77] P. Ramanan and K. Nakayama, 'Baffle : Blockchain based aggregator free federated learning,' in *2020 IEEE International Conference on Blockchain (Blockchain)*, 2020, pp. 72–81. DOI: 10.1109/Blockchain50366.2020.00017.

[78] X. Chen, J. Ji, C. Luo, W. Liao and P. Li, 'When machine learning meets blockchain: A decentralized, privacy-preserving and secure design,' in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 1178–1187. DOI: 10.1109/BigData.2018.8622598.

[79] H. Kim, J. Park, M. Bennis and S.-L. Kim, 'Blockchained on-device federated learning,' *IEEE Communications Letters*, vol. 24, no. 6, pp. 1279–1283, 2020. DOI: 10.1109/LCOMM.2019.2921755.

[80] U. Majeed and C. S. Hong, 'Flchain: Federated learning via mec-enabled blockchain network,' in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2019, pp. 1–4. DOI: 10.23919/APNOMS.2019.8892848.

[81] H. Chai, S. Leng, Y. Chen and K. Zhang, 'A hierarchical blockchain-enabled federated learning algorithm for knowledge sharing in internet of vehicles,' *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 3975–3986, 2021. DOI: 10.1109/TITS.2020.3002712.

[82] A. P. Kalapaaking, I. Khalil and M. Atiquzzaman, 'Blockchain-enabled and multisignature-powered verifiable model for securing federated learning systems,' *IEEE Internet of Things Journal*, vol. 10, no. 24, pp. 21 410–21 420, 2023. DOI: 10.1109/JIOT.2023.3289832.

[83] A. B. Kurtulmus and K. Daniel, *Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain*, 2018. arXiv: 1802.10185 [cs.CR].

[84] T. Rückel, J. Sedlmeir and P. Hofmann, 'Fairness, integrity, and privacy in a scalable blockchain-based federated learning system,' *Computer Networks*, vol. 202, p. 108 621, 2022, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.108621. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128621005132.

[85] Z. Peng, J. Xu, X. Chu *et al.*, 'Vfchain: Enabling verifiable and auditable federated learning via blockchain systems,' *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 173–186, 2022. DOI: 10.1109/TNSE.2021.3050781.

[86] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang and W. Luo, 'Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive,' *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2438–2455, 2021. DOI: 10.1109/TDSC.2019.2952332.

[87] S. Ma, Y. Cao and L. Xiong, 'Transparent contribution evaluation for secure federated learning on blockchain,' in *2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW)*, 2021, pp. 88–91. DOI: 10.1109/ICDEW53142.2021.00023.

[88] Z. Zhang, D. Dong, Y. Ma *et al.*, 'Refiner: A reliable incentive-driven federated learning system powered by blockchain,' *Proc. VLDB Endow.*, vol. 14, no. 12, pp. 2659–2662, Jul. 2021, ISSN: 2150-8097. DOI: 10.14778/3476311.3476313. [Online]. Available: https://doi.org/10.14778/3476311.3476313.

[89] A. Fadaeddini, B. Majidi and M. Eshghi, 'Secure decentralized peer-to-peer training of deep neural networks based on distributed ledger technology,' *J. Supercomput.*, vol. 76, no. 12, pp. 10 354–10 368, Dec. 2020, ISSN: 0920-8542. DOI: 10.1007/s11227-020-03251-9. [Online]. Available: https://doi.org/10.1007/s11227-020-03251-9.

[90] X. Bao, C. Su, Y. Xiong, W. Huang and Y. Hu, 'Flchain: A blockchain for auditable federated learning with trust and incentive,' in *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*, 2019, pp. 151–159. DOI: 10.1109/BIGCOM.2019.00030.

[91] V. Mothukuri, R. M. Parizi, S. Pouriyeh, A. Dehghantanha and K.-K. R. Choo, 'Fabricfl: Blockchain-in-the-loop federated learning for trusted decentralized systems,' *IEEE Systems Journal*, vol. 16, no. 3, pp. 3711–3722, 2022. DOI: 10.1109/JSYST.2021.3124513.

[92] L. Ouyang, Y. Yuan and F.-Y. Wang, 'Learning markets: An ai collaboration framework based on blockchain and smart contracts,' *IEEE Internet of Things Journal*, vol. 9, no. 16, pp. 14 273–14 286, 2022. DOI: 10.1109/JIOT.2020.3032706.

[93] M. H. ur Rehman, K. Salah, E. Damiani and D. Svetinovic, 'Towards blockchain-based reputation-aware federated learning,' in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2020, pp. 183–188. DOI: 10.1109/INFOCOMWKSHPS50562.2020.9163027.

[94] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang and M. Guizani, 'Reliable federated learning for mobile networks,' *IEEE Wireless Communications*, vol. 27, no. 2, pp. 72–80, 2020. DOI: 10.1109/MWC.001.1900119.

[95] M. Shayan, C. Fung, C. J. M. Yoon and I. Beschastnikh, 'Biscotti: A blockchain system for private and secure federated learning,' *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1513–1525, 2021. DOI: 10.1109/TPDS.2020.3044223.

[96] V. Shejwalkar and A. Houmansadr, 'Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning,' in *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium*, 2021.

[97] P. Blanchard, E. M. El Mhamdi, R. Guerraoui and J. Stainer, 'Machine learning with adversaries: Byzantine tolerant gradient descent,' in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf.

[98] D. Yin, Y. Chen, R. Kannan and P. Bartlett, 'Byzantine-robust distributed learning: Towards optimal statistical rates,' in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Jul. 2018, pp. 5650–5659. [Online]. Available: https://proceedings.mlr.press/v80/yin18a.html.

[99] E. M. El Mhamdi, R. Guerraoui and S. Rouault, 'The hidden vulnerability of distributed learning in Byzantium,' in *Proceedings of the 35th International Conference on*

Machine Learning, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Jul. 2018, pp. 3521–3530. [Online]. Available: https://proceedings.mlr.press/v80/mhamdi18a.html.

[100] Y. Li, A. S. Sani, D. Yuan and W. Bao, 'Enhancing federated learning robustness through clustering non-iid features,' in *Proceedings of the Asian Conference on Computer Vision (ACCV) Workshops*, Dec. 2022, pp. 41–55.

[101] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang and H. Qi, 'Beyond inferring class representatives: User-level privacy leakage from federated learning,' in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, Paris, France: IEEE Press, 2019, pp. 2512–2520. DOI: 10.1109/INFOCOM.2019.8737416. [Online]. Available: https://doi.org/10.1109/INFOCOM.2019.8737416.

[102] L. Melis, C. Song, E. De Cristofaro and V. Shmatikov, 'Exploiting unintended feature leakage in collaborative learning,' in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 691–706. DOI: 10.1109/SP.2019.00029.

[103] J. Geiping, H. Bauermeister, H. Dröge and M. Moeller, 'Inverting gradients - how easy is it to break privacy in federated learning?' In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 16 937–16 947. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/c4ede56bbd98819ae6112b20ac6bf145-Paper.pdf.

[104] B. Balle, G. Cherubin and J. Hayes, 'Reconstructing training data with informed adversaries,' in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 1138–1156. DOI: 10.1109/SP46214.2022.9833677.

[105] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz and P. Molchanov, 'See through gradients: Image batch recovery via gradinversion,' in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 16 337–16 346.

[106] Z. He, T. Zhang and R. B. Lee, 'Model inversion attacks against collaborative inference,' in *Proceedings of the 35th Annual Computer Security Applications Conference*, ser. ACSAC '19, San Juan, Puerto Rico, USA: Association for Computing Machinery,

2019, pp. 148–162, ISBN: 9781450376280. DOI: 10.1145/3359789.3359824. [Online]. Available: https://doi.org/10.1145/3359789.3359824.

[107] E. Erdoğan, A. Küpçü and A. E. Çiçek, 'Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning,' in *Proceedings of the 21st Workshop on Privacy in the Electronic Society*, ser. WPES'22, Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 115–124, ISBN: 9781450398732. DOI: 10.1145/3559613.3563201. [Online]. Available: https://doi.org/10.1145/3559613.3563201.

[108] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino and N. Kourtellis, 'Ppfl: Privacy-preserving federated learning with trusted execution environments,' in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '21, Virtual Event, Wisconsin: Association for Computing Machinery, 2021, pp. 94–108, ISBN: 9781450384438. DOI: 10.1145/3458864.3466628. [Online]. Available: https://doi.org/10.1145/3458864.3466628.

[109] P. Vepakomma, A. Singh, O. Gupta and R. Raskar, *Nopeek: Information leakage reduction to share activations in distributed deep learning*, 2020. DOI: 10.1109/ICDMW51313.2020.00134.

[110] D. Pasquini, G. Ateniese and M. Bernaschi, 'Unleashing the tiger: Inference attacks on split learning,' in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21, Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 2113–2129, ISBN: 9781450384544. DOI: 10.1145/3460120.3485259. [Online]. Available: https://doi.org/10.1145/3460120.3485259.

[111] B. Hitaj, G. Ateniese and F. Perez-Cruz, 'Deep models under the gan: Information leakage from collaborative deep learning,' in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 603–618, ISBN: 9781450349468. DOI: 10.1145/3133956.3134012. [Online]. Available: https://doi.org/10.1145/3133956.3134012.

[112] M. Song, Z. Wang, Z. Zhang *et al.*, 'Analyzing user-level privacy attack against federated learning,' *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2430–2444, 2020. DOI: 10.1109/JSAC.2020.3000372.

[113] O. Li, J. Sun, X. Yang *et al.*, 'Label leakage and protection in two-party split learning,' in *International Conference on Learning Representations*, 2022.

[114] S. Yuan, M. Xue, K. Wang and M. Shen, 'Deep gradient attack with strong dp-sgd lower bound for label privacy,' in *Proc. Workshop Secur. Saf. Mach. Learn. Syst.(ICLR)*, 2021.

[115] S. Kariyappa and M. K. Qureshi, 'Exploit: Extracting private labels in split learning,' in *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2023, pp. 165–175. DOI: 10.1109/SaTML54575.2023.00020.

[116] S. Xie, X. Yang, Y. Yao, T. Liu, T. Wang and J. Sun, *Label inference attack against split learning under regression setting*, 2023. arXiv: 2301.07284 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2301.07284.

[117] J. Liu, X. Lyu, Q. Cui and X. Tao, 'Similarity-based label inference attack against training and inference of split learning,' *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 2881–2895, 2024. DOI: 10.1109/TIFS.2024.3356821.

[118] X. Luo, Y. Wu, X. Xiao and B. C. Ooi, 'Feature inference attack on model predictions in vertical federated learning,' in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 181–192. DOI: 10.1109/ICDE51399.2021.00023.

[119] M. Nasr, R. Shokri and A. Houmansadr, 'Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning,' in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 739–753. DOI: 10.1109/SP.2019.00065.

[120] R. Shokri, M. Stronati, C. Song and V. Shmatikov, 'Membership inference attacks against machine learning models,' in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 3–18. DOI: 10.1109/SP.2017.41.

[121] X. Yin, Y. Zhu and J. Hu, 'A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions,' *ACM Comput. Surv.*, vol. 54,

no. 6, Jul. 2021, ISSN: 0360-0300. DOI: 10.1145/3460427. [Online]. Available: https://doi.org/10.1145/3460427.

[122] P. Paillier, 'Public-key cryptosystems based on composite degree residuosity classes,' in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238, ISBN: 978-3-540-48910-8.

[123] X. Yi, R. Paulet and E. Bertino, 'Homomorphic encryption,' in *Homomorphic Encryption and Applications*. Cham: Springer International Publishing, 2014, pp. 27–46, ISBN: 978-3-319-12229-8. DOI: 10.1007/978-3-319-12229-8_2. [Online]. Available: https://doi.org/10.1007/978-3-319-12229-8_2.

[124] L. T. Phong, Y. Aono, T. Hayashi, L. Wang and S. Moriai, 'Privacy-preserving deep learning via additively homomorphic encryption,' *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018. DOI: 10.1109/TIFS.2017.2787987.

[125] J. Ma, S.-A. Naas, S. Sigg and X. Lyu, 'Privacy-preserving federated learning based on multi-key homomorphic encryption,' *International Journal of Intelligent Systems*, vol. 37, no. 9, pp. 5880–5901, 2022. DOI: https://doi.org/10.1002/int.22818. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/int.22818. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/int.22818.

[126] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan and Y. Liu, 'Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning,' in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC'20, USA: USENIX Association, 2020, ISBN: 978-1-939133-14-4.

[127] H. Fang and Q. Qian, 'Privacy preserving machine learning with homomorphic encryption and federated learning,' *Future Internet*, vol. 13, no. 4, 2021, ISSN: 1999-5903. DOI: 10.3390/fi13040094. [Online]. Available: https://www.mdpi.com/1999-5903/13/4/94.

[128] G. Xu, X. Han, S. Xu *et al.*, 'Hercules: Boosting the performance of privacy-preserving federated learning,' *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 5, pp. 4418–4433, 2023. DOI: 10.1109/TDSC.2022.3218793.

[129] Y. Li, Y. Zhou, A. Jolfaei, D. Yu, G. Xu and X. Zheng, 'Privacy-preserving federated learning framework based on chained secure multiparty computing,' *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6178–6186, 2021. DOI: 10.1109/JIOT.2020.3022911.

[130] K. Bonawitz, V. Ivanov, B. Kreuter *et al.*, 'Practical secure aggregation for privacy-preserving machine learning,' in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1175–1191, ISBN: 9781450349468. DOI: 10.1145/3133956.3133982. [Online]. Available: https://doi.org/10.1145/3133956.3133982.

[131] B. Choi, J.-y. Sohn, D.-J. Han and J. Moon, *Communication-computation efficient secure aggregation for federated learning*, 2021. arXiv: 2012.05433 [cs.LG].

[132] J. So, B. Güler and A. S. Avestimehr, 'Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning,' *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 1, pp. 479–489, 2021. DOI: 10.1109/JSAIT.2021.3054610.

[133] Y. Li, Y. Zhou, A. Jolfaei, D. Yu, G. Xu and X. Zheng, 'Privacy-preserving federated learning framework based on chained secure multiparty computing,' *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6178–6186, 2021. DOI: 10.1109/JIOT.2020.3022911.

[134] A. R. Elkordy and A. S. Avestimehr, 'Heterosag: Secure aggregation with heterogeneous quantization in federated learning,' *IEEE Transactions on Communications*, vol. 70, no. 4, pp. 2372–2386, 2022. DOI: 10.1109/TCOMM.2022.3151126.

[135] J. So, R. E. Ali, B. Güler, J. Jiao and A. S. Avestimehr, 'Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning,' *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 8, pp. 9864–9873, Jun. 2023. DOI: 10.1609/aaai.v37i8.26177. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/26177.

[136] J. Zhang, X. Li, K. Gu, W. Liang and K. Li, 'Secure aggregation in heterogeneous federated learning for digital ecosystems,' *IEEE Transactions on Consumer Electronics*, pp. 1–1, 2023. DOI: 10.1109/TCE.2023.3330501.

[137] Y. Zheng, S. Lai, Y. Liu, X. Yuan, X. Yi and C. Wang, 'Aggregation service for federated learning: An efficient, secure, and more resilient realization,' *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 988–1001, 2023. DOI: 10.1109/TDSC.2022.3146448.

[138] L. Yin, J. Feng, H. Xun, Z. Sun and X. Cheng, 'A privacy-preserving federated learning for multiparty data sharing in social iots,' *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2706–2718, 2021. DOI: 10.1109/TNSE.2021.3074185.

[139] C. Wang, B. Lin, N. Liu, Z. Liu, J. Zhang and Q. Gan, 'Ring-ppfl: A ring topology based framework for privacy-preserving federated learning,' in *2024 7th World Conference on Computing and Communication Technologies (WCCCT)*, 2024, pp. 37–42. DOI: 10.1109/WCCCT60665.2024.10541670.

[140] 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. DOI: 10.1109/TIFS.2023.3243490.

[141] J. Li, A. S. Rakin, X. Chen, Z. He, D. Fan and C. Chakrabarti, 'Ressfl: A resistance transfer framework for defending model inversion attack in split federated learning,' in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 10 184–10 192. DOI: 10.1109/CVPR52688.2022.00995.

[142] Y. Mao, Z. Xin, Z. Li, J. Hong, Q. Yang and S. Zhong, *Secure split learning against property inference, data reconstruction, and feature space hijacking attacks*, 2023. arXiv: 2304.09515 [cs.LG].

[143] H. B. McMahan, D. Ramage, K. Talwar and L. Zhang, 'Learning differentially private recurrent language models,' in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=BJ0hF1Z0b.

[144] M. Wu, G. Cheng, P. Li *et al.*, 'Split learning with differential privacy for integrated terrestrial and non-terrestrial networks,' *IEEE Wireless Communications*, vol. 31, no. 3, pp. 177–184, 2024. DOI: 10.1109/MWC.015.2200462.

[145] A. Iqbal, P. Gope and B. Sikdar, *Privacy-preserving collaborative split learning framework for smart grid load forecasting*, 2024. arXiv: 2403.01438 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2403.01438.

[146] X. Yang, J. Sun, Y. Yao, J. Xie and C. Wang, *Differentially private label protection in split learning*, 2022. arXiv: 2203.02073 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2203.02073.

[147] A. Girgis, D. Data, S. Diggavi, P. Kairouz and A. Theertha Suresh, 'Shuffled model of differential privacy in federated learning,' in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, A. Banerjee and K. Fukumizu, Eds., ser. Proceedings of Machine Learning Research, vol. 130, PMLR, Apr. 2021, pp. 2521–2529. [Online]. Available: https://proceedings.mlr.press/v130/girgis21a.html.

[148] A. Cheu, A. Smith, J. Ullman, D. Zeber and M. Zhilyaev, *Distributed differential privacy via shuffling*, Cryptology ePrint Archive, Paper 2019/245, 2019. [Online]. Available: https://eprint.iacr.org/2019/245.

[149] X. Liu, H. Li, G. Xu, R. Lu and M. He, 'Adaptive privacy-preserving federated learning,' *Peer-to-peer networking and applications*, vol. 13, pp. 2356–2366, 2020.

[150] M. Talaei and I. Izadi, *Adaptive differential privacy in federated learning: A priority-based approach*, 2022. [Online]. Available: https://openreview.net/forum?id=FVJTyOUJzti.

[151] S. Truex, N. Baracaldo, A. Anwar *et al.*, 'A hybrid approach to privacy-preserving federated learning,' in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, ser. AISec'19, London, United Kingdom: Association for Computing Machinery, 2019, pp. 1–11, ISBN: 9781450368339. DOI: 10.1145/3338501.3357370. [Online]. Available: https://doi.org/10.1145/3338501.3357370.

[152] W. Mou, C. Fu, Y. Lei and C. Hu, 'A verifiable federated learning scheme based on secure multi-party computation,' in *Wireless Algorithms, Systems, and Applications: 16th International Conference, WASA 2021, Nanjing, China, June 25–27, 2021, Proceedings, Part II*, Nanjing, China: Springer-Verlag, 2021, pp. 198–209, ISBN: 978-3-030-86129-2. DOI: 10.1007/978-3-030-86130-8_16. [Online]. Available: https://doi.org/10.1007/978-3-030-86130-8_16.

[153] D. Byrd and A. Polychroniadou, 'Differentially private secure multi-party computation for federated learning in financial applications,' in *Proceedings of the First ACM International Conference on AI in Finance*, ser. ICAIF '20, New York, New York: Association for Computing Machinery, 2021, ISBN: 9781450375849. DOI: 10.1145/3383455.3422562. [Online]. Available: https://doi.org/10.1145/3383455.3422562.

[154] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar and H. Ludwig, 'Hybridalpha: An efficient approach for privacy-preserving federated learning,' in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, ser. AISec'19, London, United Kingdom: Association for Computing Machinery, 2019, pp. 13–23, ISBN: 9781450368339. DOI: 10.1145/3338501.3357371. [Online]. Available: https://doi.org/10.1145/3338501.3357371.

[155] A. Hosna, E. Merry, J. Gyalmo, Z. Alom, Z. Aung and M. A. Azim, 'Transfer learning: A friendly introduction,' *Journal of Big Data*, vol. 9, no. 1, p. 102, 2022.

[156] X. Liu, W. Yu, F. Liang, D. Griffith and N. Golmie, 'Toward deep transfer learning in industrial internet of things,' *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12 163–12 175, 2021. DOI: 10.1109/JIOT.2021.3062482.

[157] S. Tammina, 'Transfer learning using vgg-16 with deep convolutional neural network for classifying images,' *International Journal of Scientific and Research Publications (IJSRP)*, vol. 9, no. 10, p9420, 2019.

[158] C. Iorga and V.-E. Neagoe, 'A deep cnn approach with transfer learning for image recognition,' in *2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 2019, pp. 1–6. DOI: 10.1109/ECAI46879.2019.9042173.

[159] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing and R. Feris, 'Spottune: Transfer learning through adaptive fine-tuning,' in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4800–4809. DOI: 10.1109/CVPR.2019.00494.

[160] S. Han, J. Pool, J. Tran and W. J. Dally, 'Learning both weights and connections for efficient neural networks,' in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 1135–1143.

[161] C. Gamanayake, L. Jayasinghe, B. K. K. Ng and C. Yuen, 'Cluster pruning: An efficient filter pruning method for edge ai vision applications,' *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 802–816, 2020. DOI: 10.1109/JSTSP.2020.2971418.

[162] Z. Chang, S. Liu, X. Xiong, Z. Cai and G. Tu, 'A survey of recent advances in edge-computing-powered artificial intelligence of things,' *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13 849–13 875, 2021. DOI: 10.1109/JIOT.2021.3088875.

[163] S. Anwar, K. Hwang and W. Sung, 'Structured pruning of deep convolutional neural networks,' *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, Feb. 2017, ISSN: 1550-4832. DOI: 10.1145/3005348. [Online]. Available: https://doi.org/10.1145/3005348.

[164] S. Scardapane, M. Scarpiniti, E. Baccarelli and A. Uncini, 'Why should we add early exits to neural networks?' *Cognitive Computation*, vol. 12, no. 5, pp. 954–966, 2020.

[165] S. Laskaridis, A. Kouris and N. D. Lane, 'Adaptive inference through early-exit networks: Design, challenges and directions,' in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, ser. EMDL'21, Virtual, WI, USA: Association for Computing Machinery, 2021, pp. 1–6, ISBN: 9781450385978. DOI: 10.1145/3469116.3470012. [Online]. Available: https://doi.org/10.1145/3469116.3470012.

[166] Y. Bengio, P. Lamblin, D. Popovici and H. Larochelle, 'Greedy layer-wise training of deep networks,' in *Advances in Neural Information Processing Systems*, B. Schölkopf, J. Platt and T. Hoffman, Eds., vol. 19, MIT Press, 2006. [Online]. Available: https:

//proceedings.neurips.cc/paper_files/paper/2006/file/5da713a690c067105aeb2fae32403405-Paper.pdf.

[167] E. Belilovsky, M. Eickenberg and E. Oyallon, 'Greedy layerwise learning can scale to ImageNet,' in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Jun. 2019, pp. 583–593. [Online]. Available: https://proceedings.mlr.press/v97/belilovsky19a.html.

[168] S. Teerapittayanon, B. McDanel and H. T. Kung, 'Branchynet: Fast inference via early exiting from deep neural networks,' *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, 2016.

[169] Q. Zhang, C. Xin and H. Wu, 'Gala: Greedy computation for linear algebra in privacy-preserved neural networks,' in *Network and Distributed Systems Security (NDSS) Symposium*, 2021.

[170] Y. Kaya, S. Hong and T. Dumitras, 'Shallow-deep networks: Understanding and mitigating network overthinking,' in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Jun. 2019, pp. 3301–3310. [Online]. Available: https://proceedings.mlr.press/v97/kaya19a.html.

[171] L. Zhang, C. Bao and K. Ma, 'Self-distillation: Towards efficient and compact neural networks,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 4388–4403, 2022. DOI: 10.1109/TPAMI.2021.3067100.

[172] P. Garcia Lopez, A. Montresor, D. Epema *et al.*, 'Edge-centric computing: Vision and challenges,' *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, Sep. 2015, ISSN: 0146-4833. DOI: 10.1145/2831347.2831354. [Online]. Available: https://doi.org/10.1145/2831347.2831354.

[173] I. Ud Din, M. Guizani, B.-S. Kim, S. Hassan and M. Khurram Khan, 'Trust management techniques for the internet of things: A survey,' *IEEE Access*, vol. 7, pp. 29 763–29 787, 2019. DOI: 10.1109/ACCESS.2018.2880838.

[174] H.-N. Dai, Z. Zheng and Y. Zhang, 'Blockchain for internet of things: A survey,' *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, 2019. DOI: 10.1109/JIOT.2019.2920987.

[175] R. Yang, F. R. Yu, P. Si, Z. Yang and Y. Zhang, 'Integrated blockchain and edge computing systems: A survey, some research issues and challenges,' *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1508–1532, 2019. DOI: 10.1109/COMST.2019.2894727.

[176] S. D. Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri and V. Sassone, 'Pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain,' in *Italian Conference on Cybersecurity*, 2018.

[177] S. Kaur, S. Chaturvedi, A. Sharma and J. Kar, 'A research survey on applications of consensus protocols in blockchain,' *Security and Communication Networks*, vol. 2021, no. 1, p. 6 693 731, 2021. DOI: https://doi.org/10.1155/2021/6693731. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/6693731. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/6693731.

[178] J. de Carvalho Silva, J. J. P. C. Rodrigues, A. M. Alberti, P. Solic and A. L. L. Aquino, 'Lorawan — a low power wan protocol for internet of things: A review and opportunities,' in *2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*, 2017, pp. 1–6.

[179] J. Haxhibeqiri, E. De Poorter, I. Moerman and J. Hoebeke, 'A survey of lorawan for iot: From technology to application,' *Sensors*, vol. 18, no. 11, 2018, ISSN: 1424-8220. DOI: 10.3390/s18113995. [Online]. Available: https://www.mdpi.com/1424-8220/18/11/3995.

[180] S. Caldas, S. M. K. Duddu, P. Wu *et al.*, *Leaf: A benchmark for federated settings*, 2019. arXiv: 1812.01097 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1812.01097.

[181] G. Cohen, S. Afshar, J. Tapson and A. van Schaik, *Emnist: An extension of mnist to handwritten letters*, 2017. arXiv: 1702.05373 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1702.05373.

[182] A. Krizhevsky, G. Hinton *et al.*, 'Learning multiple layers of features from tiny images,' 2009.

[183] Z. Zhang, A. Pinto, V. Turina, F. Esposito and I. Matta, 'Privacy and efficiency of communications in federated split learning,' *IEEE Transactions on Big Data*, vol. 9, no. 5, pp. 1380–1391, 2023. DOI: 10.1109/TBDATA.2023.3280405.

[184] N. D. Pham, A. Abuadbba, Y. Gao, K. T. Phan and N. Chilamkurti, 'Binarizing split learning for data privacy enhancement and computation reduction,' *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3088–3100, 2023. DOI: 10.1109/TIFS.2023.3274391.

[185] T. Khan, K. Nguyen and A. Michalas, 'A more secure split: Enhancing the security of privacy-preserving split learning,' in *Secure IT Systems: 28th Nordic Conference, NordSec 2023, Oslo, Norway, November 16–17, 2023, Proceedings*, Oslo, Norway: Springer-Verlag, 2023, pp. 307–329, ISBN: 978-3-031-47747-8. DOI: 10.1007/978-3-031-47748-5_17. [Online]. Available: https://doi.org/10.1007/978-3-031-47748-5_17.

[186] S. A. Khowaja, I. H. Lee, K. Dev, M. A. Jarwar and N. M. F. Qureshi, 'Get your foes fooled: Proximal gradient split learning for defense against model inversion attacks on iomt data,' *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 5, pp. 2607–2616, 2023. DOI: 10.1109/TNSE.2022.3188575.

[187] R. Shwartz-Ziv and N. Tishby, 'Opening the black box of deep neural networks via information,' *Information Flow in Deep Neural Networks*, p. 24, 2022.

[188] Z. Goldfeld, E. Van Den Berg, K. Greenewald *et al.*, 'Estimating information flow in deep neural networks,' in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Jun. 2019, pp. 2299–2308. [Online]. Available: https://proceedings.mlr.press/v97/goldfeld19a.html.

[189] C. Thapa, M. A. P. Chamikara, S. Camtepe and L. Sun, 'Splitfed: When federated learning meets split learning,' 2022. arXiv: 2004.12088 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2004.12088.

[190] J. Konečný, H. B. McMahan, D. Ramage and P. Richtárik, *Federated optimization: Distributed machine learning for on-device intelligence*, 2016. arXiv: 1610.02527 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1610.02527.

[191] Y. LeCun, B. Boser, J. S. Denker *et al.*, 'Backpropagation applied to handwritten zip code recognition,' *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[192] S. M. Lundberg and S.-I. Lee, 'A unified approach to interpreting model predictions,' in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 4768–4777, ISBN: 9781510860964.

[193] K. Ganju, Q. Wang, W. Yang, C. A. Gunter and N. Borisov, 'Property inference attacks on fully connected neural networks using permutation invariant representations,' in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18, Toronto, Canada: Association for Computing Machinery, 2018, pp. 619–633, ISBN: 9781450356930. DOI: 10.1145/3243734.3243834. [Online]. Available: https://doi.org/10.1145/3243734.3243834.

[194] M. P. M. Parisot, B. Pejo and D. Spagnuelo, *Property inference attacks on convolutional neural networks: Influence and implications of target model's complexity*, 2021. arXiv: 2104.13061 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2104.13061.

[195] Z. Wang, Y. Huang, M. Song, L. Wu, F. Xue and K. Ren, 'Poisoning-assisted property inference attack against federated learning,' *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 3328–3340, 2023. DOI: 10.1109/TDSC.2022.3196646.

[196] X. Ma, B. Li, Q. Jiang, Y. Chen, S. Gao and J. Ma, 'Nosnoop: An effective collaborative meta-learning scheme against property inference attack,' *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6778–6789, 2022. DOI: 10.1109/JIOT.2021.3112737.

[197] D. Gopinath, H. Converse, C. Pasareanu and A. Taly, 'Property inference for deep neural networks,' in *2019 34th IEEE/ACM International Conference on Automated*

Software Engineering (ASE), 2019, pp. 797–809. DOI: 10.1109/ASE.2019.00079.

[198] S. Mahloujifar, E. Ghosh and M. Chase, 'Property inference from poisoning,' in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 1120–1137. DOI: 10.1109/SP46214.2022.9833623.

[199] D. Bau, J.-Y. Zhu, H. Strobelt, A. Lapedriza, B. Zhou and A. Torralba, 'Understanding the role of individual units in a deep neural network,' *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30 071–30 078, 2020. DOI: 10.1073/pnas.1907375117. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1907375117. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.1907375117.

[200] I. Goodfellow, J. Pouget-Abadie, M. Mirza *et al.*, 'Generative adversarial nets,' *Advances in neural information processing systems*, vol. 27, 2014.

[201] K. Pearson and F. Galton, 'Vii. note on regression and inheritance in the case of two parents,' *Proceedings of the Royal Society of London*, vol. 58, no. 347-352, pp. 240–242, 1895. DOI: 10.1098/rspl.1895.0041. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspl.1895.0041. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rspl.1895.0041.

[202] S. Kullback and R. A. Leibler, 'On information and sufficiency,' *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951, ISSN: 00034851. [Online]. Available: http://www.jstor.org/stable/2236703 (visited on 11/06/2024).

[203] F. Perez-Cruz, 'Kullback-leibler divergence estimation of continuous distributions,' in *2008 IEEE International Symposium on Information Theory*, 2008, pp. 1666–1670. DOI: 10.1109/ISIT.2008.4595271.

[204] T. Caliński and J. Harabasz, 'A dendrite method for cluster analysis,' *Communications in Statistics*, vol. 3, no. 1, pp. 1–27, 1974. DOI: 10.1080/03610927408827101.

[205] D. L. Davies and D. W. Bouldin, 'A cluster separation measure,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, 1979. DOI: 10.1109/TPAMI.1979.4766909.

[206] B. C. Ross, 'Mutual information between discrete and continuous data sets,' *PLOS ONE*, vol. 9, no. 2, Feb. 2014. DOI: 10.1371/journal.pone.0087357. [Online]. Available: https://doi.org/10.1371/journal.pone.0087357.

[207] Z. He, T. Zhang and R. B. Lee, 'Attacking and protecting data privacy in edge–cloud collaborative inference systems,' *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9706–9716, 2021. DOI: 10.1109/JIOT.2020.3022358.

[208] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis and N. D. Lane, 'Spinn: Synergistic progressive inference of neural networks over device and cloud,' in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '20, London, United Kingdom: Association for Computing Machinery, 2020, ISBN: 9781450370851. DOI: 10.1145/3372224.3419194. [Online]. Available: https://doi.org/10.1145/3372224.3419194.

[209] Y. Matsubara, D. Callegaro, S. Singh, M. Levorato and F. Restuccia, 'Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing,' in *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2022, pp. 337–346. DOI: 10.1109/WoWMoM54355.2022.00032.

[210] S. Yao, J. Li, D. Liu *et al.*, 'Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency,' in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, ser. SenSys '20, Virtual Event, Japan: Association for Computing Machinery, 2020, pp. 476–488, ISBN: 9781450375900. DOI: 10.1145/3384419.3430898. [Online]. Available: https://doi.org/10.1145/3384419.3430898.

[211] B. Shah and H. Bhavsar, 'Time complexity in deep learning models,' *Procedia Computer Science*, vol. 215, pp. 202–210, 2022, 4th International Conference on Innovative Data Communication Technology and Application, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2022.12.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050922020944.

[212] N. Moustafa and J. Slay, 'Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set),' in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

[213] J. Frankle and M. Carbin, *The lottery ticket hypothesis: Finding sparse, trainable neural networks*, 2019. arXiv: 1803.03635 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1803.03635.

[214] H. Huang, J. Lin, B. Zheng, Z. Zheng and J. Bian, 'When blockchain meets distributed file systems: An overview, challenges, and open issues,' *IEEE Access*, vol. 8, pp. 50 574–50 586, 2020. DOI: 10.1109/ACCESS.2020.2979881.