# Advanced Machine Learning for Privacy-Preserving Intrusion Detection in IoT Networks

**Duc Manh Bui**

Thesis submitted in fulfilment of the requirements for the degree of
*Master of Science (Research) in Computing Science*

under the supervision of
A/Prof. Hoang Dinh
A/Prof. Diep N. Nguyen

School of Electrical and Data Engineering
Faculty of Engineering and Information Technology
University of Technology Sydney

February 2025

# CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Duc Manh Bui, declare that this thesis is submitted in fulfilment of the requirements for the degree of Master of Science (Research) in Computing Science at the Faculty of Engineering and Information Technology, University of Technology Sydney.
This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis This document has not been submitted for qualifications at any other academic institution.
This research is supported by the Australian Government Research Training Program.

Duc Manh Bui
February 2025

# Acknowledgements

First and foremost, I would like to extend my deepest gratitude to my supervisor, Assoc. Professor Hoang Dinh, whose encouragement to embark on this journey and invaluable insights have been instrumental to the completion of this work. Without his guidance and support, reaching this milestone would not have been possible. I am also profoundly grateful to my co-supervisor, Assoc. Professor Diep Nguyen, for his critical and constructive feedback, which has greatly improved my research skills and enhanced the quality of this thesis. I am also grateful to my family, my beloved partner My-Linh Ho, and my friends at UTS for their love and support.

# Abstract


## Advanced Machine Learning for Privacy-Preserving Intrusion Detection in IoT Networks

The Internet of Things (IoT) is a groundbreaking technology that integrates smart devices into communication networks, reducing human intervention and fueling innovation across various applications, from smart cities to intelligent transportation. However, its heterogeneous and highly interconnected nature introduces significant vulnerabilities, expanding attack surfaces that are increasingly targeted by sophisticated cyber threats. To address this, machine learning (ML), especially deep learning (DL) has shown the potential to equip IoT components with intelligent cyberattack detection modules, enabling IoT components to detect and respond to cyberattacks by analyzing vast data streams. However, traditional approaches that transmit vast amounts of user data to centralized models for processing pose significant data privacy risks, exposing sensitive information to potential misuse by the model's owner. This highlights the need for advanced distributed ML/DL methods that can accurately detect cyberattacks and preserve user data privacy.

Additionally, IoT network data often contains highly sensitive user information (e.g., location details and biometric data) making it susceptible to privacy breaches when being analyzed for cyberattack detection using ML models. To tackle this problem, homomorphic encryption (HE) offers a powerful technique to integrate with ML/DL models, enabling innovative privacy-preserving ML (PPML) approaches that safeguard the confidentiality of user data. By employing this cryptographic technique, ML/DL models can compute directly on ciphertext without the need for decryption. However, PPML using HE is still an emerging field, and designing HE-based learning algorithms remains a significant challenge due to the high computational overhead of HE and the complexity of ML/DL models. Therefore, the design of HE-based PPML algorithms to balance security, accuracy, and performance is crucial for practical deployment in real-world IoT systems.

In this thesis, we contribute to addressing both security and privacy concerns in two emerging IoT ecosystems: (1) developing a novel framework that integrates federated learning (FL) and HE for privacy-preserving intrusion detection in resource-constrained Internet

of Vehicle (IoV) networks, and (2) designing a privacy-preserving cyberattack detection in blockchain-based IoT networks using our proposed deep neural network (DNN) training algorithm for HE-encrypted data and our novel privacy-preserving distributed-cloud native learning algorithm. Following the Single-Instruction-Multiple-Data (SIMD) manner, our proposed packing algorithms enable an efficient privacy-preserving learning/inference for encrypted data, achieving results nearly identical to the non-encrypted approach, approximately from 0.01 to 0.8%. These contributions demonstrate the effectiveness and potential of our research in leveraging advanced ML to tackle security challenges and ensure data confidentiality within real-world IoT ecosystems.

# List of Publications

## Journal Papers

J-1. **B. D. Manh**, C. H. Nguyen, D. T. Hoang, D. N. Nguyen, M. Zheng and Q. V. Pham, "Privacy-Preserving Cyberattack Detection in Blockchain-Based IoT Systems Using AI and Homomorphic Encryption," *IEEE Internet of Things Journal*, Jan. 2025, doi: 10.1109/JIOT.2025.3535792.

J-2. M. A. Hassan, **B. D. Manh**, C. T. Nguyen, C. H. Nguyen, D. T. Hoang, D. N. Nguyen, N. V. Huynh and D. Niyato, "SBW 3.0: A Blockchain-Enabled Framework for Secure and Efficient Information Management in Web 3.0," *IEEE Transactions on Network and Service Management* (major revision).

J-3. M. A. Hassan, M. B. Jamshidi, **B. D. Manh**, N. H. Chu, C. -H. Nguyen, N. Q. Hieu, C. T. Nguyen, D. T. Hoang, D. N. Nguyen, N. V. Huynh, M. A. Alsheikh and E. Dutkiewicz, " Enabling Technologies for Web 3.0: A Comprehensive Survey," *Elsevier Computer Networks* (major revision).

J-4. **B. D. Manh**, C. H. Nguyen, D. T. Hoang and D. N. Nguyen, "Towards Zero-Trust in IoT Networks: A Homomorphic Encryption-Enabled Federated Learning Approach," *IEEE Internet of Things Journal* (on going).

## Conference Papers

C-1. **B. D. Manh**, C. H. Nguyen, D. T. Hoang and D. N. Nguyen, "Homomorphic Encryption-Enabled Federated Learning for Privacy-Preserving Intrusion Detection in Resource-Constrained IoV Networks," *IEEE Vehicular Technology Conference*, October 2024, Washington DC (accepted).

C-2. C. H. Nguyen, **B. D. Manh**, D. T. Hoang and D. N. Nguyen, "Towards Secure AI-empowered Vehicular Networks: A Federated Learning Approach using Homomorphic Encryption," *IEEE Vehicular Technology Conference*, October 2024, Washington DC (accepted).

C-3. D. H. Son, **B. D. Manh**, T. V. Khoa, N. L. Trung, D. T. Hoang, H. T. Minh, Y. Alem and L. Q. Minh, “Semi-Supervised Learning for Anomaly Detection in Blockchain-based Supply Chains," *IEEE International Symposium on Communications and Information Technologies*, September 2024, Bangkok (accepted).

## Book Chapters

B-1. **B. D. Manh**, N. Q. Hieu, D. T. Hoang and D. N. Nguyen, “Machine Learning for Cyberattack Detection in Internet of Things Networks: An Overview,” *Elsevier Advanced Machine Learning for Cyber-Attack Detection in IoT Networks* (accepted)

B-2. N. Q. Hieu, **B. D. Manh**, D. T. Hoang and D. N. Nguyen, “Challenges and Potential Research Directions for Machine Learning-based Cyberattack Detection in IoT Networks,” *Elsevier Advanced Machine Learning for Cyber-Attack Detection in IoT Networks* (accepted)

# Table of contents

**List of Publications** **vi**

**List of figures** **x**

**Abbreviation** **xii**

**1 Motivation, Background and Literature Review** **1**

1.1 Motivation . . . . . 1

1.2 Background . . . . . 5

1.2.1 Machine Learning and Deep Learning . . . . . 5

1.2.2 Federated Learning . . . . . 12

1.2.3 Homomorphic Encryption . . . . . 14

1.3 Literature Review and Contribution . . . . . 16

1.3.1 Privacy-Aware Federated Learning for Intrusion Detection in IoV Networks . . . . . 16

1.3.2 Privacy-Preserving Machine Learning for Cyberattack Detection in Blockchain-based IoT Networks . . . . . 19

1.4 Thesis Organization . . . . . 23

**2 Homomorphic Encryption-Enabled Federated Learning for Privacy-Preserving Intrusion Detection in Resource-Constrained IoV Networks** **25**

2.1 System Model . . . . . 26

2.2 The Deep Neural Network for Encrypted Data . . . . . 27

2.3 The Proposed FL Implementation . . . . . 28

2.4 Performance Evaluation . . . . . 31

2.4.1 Simulation Setup . . . . . 31

2.4.2 Evaluation Metrics . . . . . 32

2.4.3 Simulation Results . . . . . 33

2.5 Summary . . . . . 35

**3 Privacy-Preserving Cyberattack Detection in Blockchain-Based IoT Systems Using AI and Homomorphic Encryption 36**
3.1 System Model . . . 37
3.1.1 Overview of Blockchain-based IoT System . . . 37
3.1.2 Privacy-Preserving Cyberattack Detection in Blockchain Networks 38
3.2 The Privacy-Preserving Distributed Learning . . . 39
3.2.1 The Proposed Training Process of Deep Neural Network for HE-Encrypted Data . . . 40
3.2.2 Implementation of Distributed Cloud-Native Learning . . . 49
3.3 Performance Evaluation . . . 50
3.3.1 Simulation Setup and Evaluation Metrics . . . 50
3.3.2 Simulation Results . . . 52
3.3.3 Blockchain Nodes-enabled HE Computational Evaluation . . . 56
3.3.4 Encrypted Inference Time Analysis . . . 59
3.4 Summary . . . 60

**4 Conclusion and Future Research Directions 62**
4.1 Conclusion . . . 62
4.2 Future Research Directions . . . 63
4.2.1 Current Research Limitations . . . 63
4.2.2 Future Directions . . . 64

**References 66**

# List of figures


1.1 The integration of ML into IDS. . . . . 4
1.2 The workflow of ML methods. Basically, ML algorithms obtain knowledge through the learning phase by training with the dataset to produce the trained module. This trained module is then applied to solve real-world problems, generating output from the incoming input data. . . . . 5
1.3 Deep neural network architecture. . . . . 8
1.4 Deep autoencoder architecture. . . . . 9
1.5 Convolutional neural network architecture. . . . . 10
1.6 Recurrent neural network architecture. . . . . 11
1.7 Generative adversarial network architecture. . . . . 12
1.8 Illustration of federated learning process. . . . . 13
1.9 The structure of the thesis. . . . . 23

2.1 The proposed privacy-preserving intrusion detection framework including pre-learning and privacy-preserving learning. . . . . 26
2.2 Encrypt process of a 3x3 matrix $\boldsymbol{W}$. . . . . 28
2.3 The general flow of converting the weight and activation functions of neural network to HE dimension. . . . . 30
2.4 Convergence of privacy-preserving learning with different number of VUs. Regardless of the amount of offloaded encrypted data, the proposed EncFL consistently achieves accuracy comparable to scenarios without decryption. 32
2.5 Classification results of 2 VUs. The accuracy of all the attacks remains consistent with non-encrypted benchmarks. . . . . 34
2.6 Classification results of 3 VUs. The same trend is observed with “Reconnaissance” attack, even with the increased offloading data. . . . . 35

3.1 The proposed privacy-preserving cyberattack detection framework includes three phases: data encryption and offloading, encrypted data training, and real-time detection. The CSP operates as a network security service to ensure the security of $N$ blockchain nodes (BNs). . . . . 39
3.2 Illustration of the proposed alternative packing method, including (a) 1D packing and (b) 2D packing. The proposed methods are the pre-processing of HE, which fits the elements of data (i.e., vector and matrix) into the slots of a ciphertext in both normal (axis=0) and transpose (axis=1) manners. Therefore, this enables efficient encrypted matrix/vector multiplications during the training task. . . . . 41
3.3 Illustration of $\mathsf{SumCols}(.)$ and $\mathsf{SumRows}(.)$ algorithms on a ciphertext with multiple segments, each having $S$ slots size where each slot represents a distinct element $x_i$ [1]. . . . . 44
3.4 Implementation of encrypted feed-forward in a neural network with two layers. The weight matrix of the neural network layer is packed and encrypted based on the $k$-th order of the considered layer, regarding normal processing (axis=0) or transpose processing (axis=1). Therefore, through the HE multiplication described in (3.9), the input ciphertext, initially packed along axis=0, can alternatively fit into the encrypted neural network layers. 45
3.5 The experiment setup of CSP environment. . . . . 51
3.6 Convergence of the considered learning algorithms. . . . . 52
3.7 Execution time of offline training phase. . . . . 53
3.8 Classification results of the non-encrypted detection within the proposed HE-encrypted deep neural network. . . . . 57
3.9 Evaluation of the *Geth* blockchain node-enabled HE in different consensus mechanisms: PoW, PoA, PoS, and no mining. . . . . 58
3.10 The inference time of the detection model with encrypted samples. . . . . 60

# Abbreviation

| | |
|---|---|
| **IoT** | The Internet of Things |
| **IoV** | The Internet of Vehicles |
| **AI** | Artificial Intelligence |
| **ML** | Machine Learning |
| **DL** | Deep Learning |
| **HE** | Homomorphic Encryption |
| **PPML** | Privacy-Preserving Machine Learning |
| **SSRF** | Server-Side Request Forgery |
| **DDoS** | Distributed Denial of Service |
| **IDS** | Intrusion Detection System |
| **NIDS** | Network Intrusion Detection System |
| **KNN** | K-Nearest Neighbour |
| **MLP** | Multilayer Perceptron |
| **DT** | Decision Tree |
| **SVM** | Support Vector Machine |
| **PCA** | Principal Component Analysis |
| **t-SNE** | t-Distributed Stochastic Neighbor Embedding |
| **AE** | Autoencoder |
| **DNN** | Deep Neural Network |
| **CNN** | Convolutional Neural Network |
| **RNN** | Recurrent Neural Network |
| **DAE** | Deep Autoencoder |
| **DBN** | Deep Belief Network |
| **GAN** | Generative Adversarial Network |
| **TL** | Transfer Learning |
| **FL** | Federated Learning |
| **CKKS** | Cheon-Kim-Kim-Seon |
| **RLWE** | Ring Learning With Errors |

| | |
|---|---|
| **SIMD** | Single Instruction Multiple Data |
| **RSUs** | Roadside Units |
| **VUs** | Vehicle Users |
| **CSP** | Cloud Service Provider |

# Chapter 1

# Motivation, Background and Literature Review

In this chapter, we first introduce the motivation of this thesis. We then present an in-depth background of machine learning, deep learning, federated learning, and homomorphic encryption. After that, we conduct a comprehensive review of existing studies on machine learning for intrusion detection in IoT networks and privacy-preserving machine learning, highlighting the advantages and limitations of current approaches. Finally, the main contributions and the structure of the thesis are presented.

## 1.1 Motivation

Over the past few years, the Internet of Things (IoT) has undergone remarkable developments, significantly improving the quality of our daily lives. This emerging technology has enabled the development of various intelligent applications spanning diverse sectors, from smart transportation and smart healthcare to smart city and industrial automation [2]. Following this trend, it is recorded that around 15 billion IoT devices were deployed in 2023, and it is projected to double by 2030, reflecting the accelerating pace of IoT adoption [3]. In general, the IoT system is defined as the interconnection between a massive number of heterogeneous devices and cloud computing systems to provide intelligent services for human benefit.

However, despite the benefits of IoT, its wide range nature also introduces various cybersecurity risks. Accordingly, maintaining the security requirements across large-scale IoT environments, which operate based on the connectivity between multiple layers within diverse areas, is still challenging. In the IoT network, each layer has its own vulnerabilities that the intruders can exploit via different types of cyberattacks [4] [5]. For instance, in 2016,

Mirai exploited a massive amount of IoT devices to form a huge distributed Botnets network, which was then used to launch a massive DDoS attack against Dyn, a major DNS provider, thereby disrupting Internet access for numerous websites and services, including Twitter, Netflix, and Spotify [6]. Additionally, in 2019, AWS, a famous cloud service which manages various IoT components, faced a data breach incident caused by a Server-Side Request Forgery (SSRF) injection, exposing the personal information of approximately 100 million customers in the United States, 6 million in Canada, and 80,000 bank account numbers [7]. Moreover, in 2020, Mozi malware showed its capability to exploit weak telnet passwords and nearly a dozen unpatched vulnerabilities of various IoT devices manufactured by Netgear, Huawei, and ZTE [8].

To minimize the security concerns of IoT networks, both industry and academia are increasingly adopting Blockchain technology, widely acknowledged as a promising solution to significantly enhance the security and resilience of IoT architectures. Blockchain technology plays a crucial role in enhancing the security, transparency, and reliability of IoT systems, addressing several of the inherent challenges associated with these networks [9]. Blockchain's decentralized ledger technology ensures that data transmitted across IoT devices is immutable, meaning that once data is recorded, it cannot be altered or deleted without consensus from the network [9], [10]. This significantly reduces the risk of unauthorized data manipulation and cyberattacks. Moreover, blockchain enables secure and transparent transactions between IoT devices by providing a trusted, tamper-proof record of all activities [4]. This is particularly important for maintaining data integrity and trust in applications like smart homes, supply chain management, and industrial IoT. By eliminating the reliance on a central authority, blockchain also helps prevent single points of failure, thereby enhancing the resilience and robustness of IoT systems against network failures and malicious attacks [9]. Due to the above benefits, various industries have been actively leveraging blockchain to securely maintain IoT applications, such as IBM blockchain for supply chain applications and IOTA blockchain for healthcare data management [11],[12].

Although the integration of blockchain with IoT fosters a more secure and efficient environment for the widespread adoption of IoT technologies, it is still vulnerable to multiple cyber threats. Statistics show that from 2011 to 2023, blockchain ecosystems have endured over 1,600 cyberattacks, resulting in financial losses totally exceeding $32 billion [13]. For instance, in August 2021, Poly Network, a cross-chain protocol for blockchain applications, reported that their system had been hacked, causing over $611 million to be stolen [14]. Additionally, in March 2022, Ronin Network, an EVM-based blockchain game application, revealed that a hacker had successfully stolen multiple private keys, resulting in a loss of $614 million [14]. In terms of network security, there are also various network attacks on

the blockchain environments. For example, in April 2021, the Hotbit wallet, a blockchain cryptocurrency wallet, reported that the database of the Hotbit wallet was deleted by network attacks from hackers [13]. Besides, in September 2021, the Solana chain also reported that the system faced a network outage due to distributed denial of service (DDoS) attacks, leading to the offline of the chain for 12 hours [13]. Most recently, in June 2024, BtcTurk, a Turkish cryptocurrency exchange, suffered a network attack that impacted ten wallets containing various cryptocurrencies, resulting in a $5.3 million freeze [13], [15]. These problems reveal persistent vulnerabilities within blockchain systems that could cause serious concerns about the security and reliability of IoT-based blockchain networks.

To address the security issues caused by cyberattacks, various security measures have been explored within IoT ecosystems, with Intrusion Detection Systems (IDS) standing out as a prominent method [16]. Following that, since IoT operates mainly based on the network layer within various interconnected components, the network intrusion detection system (NIDS) is recognized as the efficient approach to safeguard the IoT network, which performs the cyberattack detection regarding network layer [5], [16], [17]. Signature-based NIDS is the most frequent cyberattack detection approach, which has been further analyzed in IoT systems. Despite its advantage in reducing the false alarm rate during cyberattack detection, the database update to deal with new attack vectors remains challenging. Besides, anomaly-based detection is also a well-known approach in NIDS. While this approach demonstrates effectiveness in identifying novel attack types, it maintains a low false positive rate in detecting each known type of attack. Specification-based detection is another frequently used method in NIDS. This approach detects intrusion based on the specific definition of components' behaviour, which is manually defined by the experts [16]. Although specification-based cyberattack detection allows fast deployment, it lacks adaptation to a different environment and efficient time. As observed, traditional NIDS face limitations in handling the dynamic and large-scale nature of IoT networks, often resulting in increased latency and reduced detection efficiency. Moreover, their reliance on predefined rules and patterns makes them less effective against sophisticated and evolving cyber threats. As a result, the diverse and heterogeneous nature of IoT systems presents substantial challenges to the effective deployment of conventional security methods for network attack detection in real-world IoT scenarios.

Recently, machine learning (ML) and deep learning (DL) have emerged as an effective approach to developing intelligent and adaptive intrusion detection systems (IDS), improving performance by learning patterns from both normal and attack data [18], [19]. As illustrated in Figure 1.1, the ML model can operate as a smart filter to detect various attack types, including previously unknown threats, making them well-suited for heterogeneous environments (e.g.,

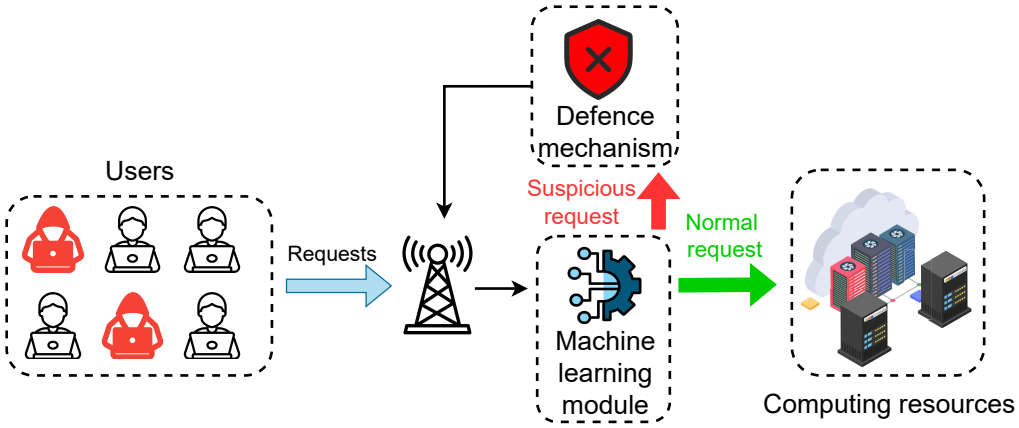



Fig. 1.1 The integration of ML into IDS.

IoT and blockchain) that frequently encounter novel cyber threats [17], [19]. Nevertheless, in IoT ecosystems, including both traditional and blockchain-based IoT, applying ML to detect cyberattacks raises substantial privacy concerns for users [20], [21]. In practice, ML-based systems typically rely on third-party services, such as cloud providers, to access the extensive computational resources required for high-demand tasks like training and inference [21]. This reliance necessitates data transfer to these third-party services, leading to potential privacy risks. While analyzing attack data, users' sensitive data, including personal details, biometric data, and healthcare records, could be extracted and accessed by these third-party providers, compromising data confidentiality [22]. Although various distributed techniques like federated learning and collaborative learning have been developed to reduce dependency on centralized training services [21], [23], they require IoT data owners to train their own data before sharing their trained models to improve overall detection performance. This approach is limited, as many IoT components (e.g., IoT gateways, IoT sensors, smartphones and wearable devices) often lack the hardware capabilities to fully train their data.

To overcome the aforementioned problems, Homomorphic Encryption (HE) offers a compelling solution, facilitating the integration with ML models to establish a robust and efficient privacy-preserving machine learning approach. By employing this lattice-based cryptographic method, data can be encrypted before being sent to third-party services, enabling ML models to perform computations directly on the encrypted data without the need for decryption [24]. This ensures that user privacy is preserved throughout the entire process. As a result, HE stands out as a highly effective enhancement for ML models, providing robust protection for users' privacy. Therefore, this thesis delves into the integration of advanced ML/DL techniques with HE to develop efficient PPML solutions tailored for IDS in two cutting-edge IoT ecosystems: the Internet of Vehicles and Blockchain-based IoT.

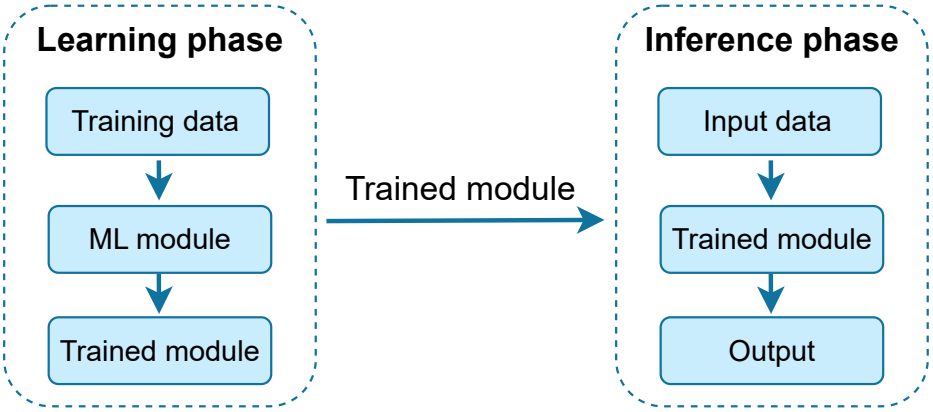



Fig. 1.2 The workflow of ML methods. Basically, ML algorithms obtain knowledge through the learning phase by training with the dataset to produce the trained module. This trained module is then applied to solve real-world problems, generating output from the incoming input data.

## 1.2 Background

### 1.2.1 Machine Learning and Deep Learning

Machine learning (ML) is a branch of artificial intelligence (AI) that empowers machines to learn particular tasks (i.e., text classifications, image classifications, automatic robot control, speech recognition, and medical diagnosis) and improve their performance based on the learnt experience [25]. In general, Machine Learning (ML) operates by automatically processing inputs and generating accurate outputs based on learned knowledge, solving decision-making problems that cannot be addressed through conventional programming written and designed by human beings. Specifically, as illustrated in Figure 1.2, the basic workflow of ML approaches is divided into two independent phases: the learning phase and the inference phase. During the learning phase, also known as the training process, the ML module acquires knowledge from the training dataset. This phase is crucial for optimizing the ML model using the training data, resulting in a trained module that is primed for deployment in the subsequent phase. Once the trained module is obtained, it is deployed to determine or predict the output based on the input data. As the training process often takes a long time to obtain the optimized ML module, it can be performed in the background while the inference phase is applied for real-time processing.

ML techniques can be broadly categorized into three main types: supervised learning, unsupervised learning, and reinforcement learning.

- *Supervised learning*: In the case of supervised learning, the ML module accesses the training phase using dataset $\mathscr{D}$ consisting of input-output pairs, where each input is

associated with a respective correct output or a label. Accordingly, the dataset can be denoted as $\mathscr{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(i)}, y^{(i)})\} \subseteq \mathbb{R}^n \times C$, in which $x^{(i)}$ is $i$-th input data sample, $y^{(i)}$ is label of $i$-th data sample, $\mathbb{R}^n$ is $n$-dimensional feature space of $x^{(i)}$ and $C$ is the label space of $y^{(i)}$. The supervised learning algorithm aims to learn the mapping from $x^{(i)}$ to $y^{(i)}$ by minimizing the difference between its predictions and actual labels during training. Hence, supervised learning is often applied to solve the classification task, in which its inference phase can identify the new input to the correct label. Regarding the binary classification (e.g., spam filtering), the label is defined as $C = \{0, 1\}$ or $C = \{-1, 1\}$. For the multiclass classification (e.g., text classification), each label is represented by an integer, with the set of possible labels being $C = \{0, 1, \ldots, K\}$, where $K$ denotes the total number of classes. Additionally, supervised learning is widely utilized to process regression tasks (e.g., weather prediction), where $C \in \mathbb{R}$.

In the training process, the dataset $\mathscr{D}$ is divided into three sets: training data $\mathscr{D}_{train}$, validation data $\mathscr{D}_{valid}$, and testing data $\mathscr{D}_{test}$. The supervised ML module strives to map the data pairs from $x^{(i)}$ to $y^{(i)}$ $(x^{(i)}, y^{(i)}) \in \mathscr{D}_{train}$ via the training function $\sigma$, which aims to generate $\sigma(x) = y$, $\big($or $\sigma(x) \approx y\big)$. Hence, a loss function, such as mean squared error or cross-entropy loss, calculates the error between the predicted probability $\sigma(x)$ and label $y$. Subsequently, the function $\sigma(.)$ is evaluated via the validation set $\mathscr{D}_{valid}$ by ML metrics (e.g., accuracy, recall, precision). To minimize the loss, ML algorithms iteratively refine $\sigma(.)$ until the ML model converges with a low error. After achieving convergence, the trained model is tested on the test set $\mathscr{D}_{test}$ to assess its overall performance and generalization capability. Regarding the supervised learning module, some well-known algorithms can be listed as follows: logistic regression, K-nearest neighbour (KNN), Bayesian classification, multilayer-perceptron (MLP), decision tree (DT), and support vector machine (SVM).

- *Unsupervised learning*: Unlike supervised approaches, unsupervised learning is a machine learning approach where models are trained on data without predefined labelled outputs, with the primary goal being to explore the hidden patterns and distribution within data. Basically, unsupervised learning uncovers relationships within the input data, primarily focusing on two tasks: clustering and dimensionality reduction. Clustering involves dividing the input data into distinct groups based on similarities among data points. On the other hand, dimensionality reduction aims to reduce the number of features in the data while preserving essential information. Some of the well-known unsupervised learning algorithms are K-means clustering, hierarchical clustering, principal component analysis (PCA), t-Distributed Stochastic Neighbor

Embedding (t-SNE), and autoencoder (AE). Due to its nature, unsupervised learning is applied in several applications, especially anomaly detection, which significantly improves the security of modern networks.

- *Reinforcement learning*: In the case of reinforcement learning, the ML module functions based on the idea of learning from consequences through trial and error without the prior dataset. Specifically, reinforcement learning enables the agent to learn by interacting and making decisions in an external environment [26]. By utilizing the reward function, the agent optimizes its actions within the environment to achieve a specific goal. This type of learning has been widely adopted in areas that require automation, such as robotics, wireless communications, and vehicles.

Despite the benefits of the aforementioned learning techniques, utilizing them with traditional ML algorithms reveals several limitations. Therefore, deep learning (DL) has been introduced as a significant solution that empowers the deep neural network (DNN) for learning and inference. DL enables the ML model to operate with a deeper network, incorporating multiple layers to enhance its performance within complex tasks. Following that, several learning approaches (e.g., semi-supervised learning, self-supervised learning and deep reinforcement learning) have been developed and seamlessly integrated with neural networks. These approaches significantly enhance the model's ability to autonomously learn complex features, reduce reliance on labelled data, and improve generalization across diverse tasks. By leveraging unlabeled or partially labelled data, self-supervised and semi-supervised learning enables more efficient representation learning, while deep reinforcement learning empowers models to make sequential decisions and adapt to dynamic environments. Accordingly, natural language processing and computer vision are two popular AI applications that leverage DL models to solve various problems (e.g., face recognition and text translation). Compared to traditional ML algorithms, DL provides several advantages, which are described as follows [25]:

- *No need to extract features manually*: To obtain the optimal solution in traditional ML algorithms, the knowledge must be well-modelled with effective feature extraction. However, it is difficult to identify the features in some specific scenarios. For instance, regarding face recognition in computer vision tasks, it is hard to generalize the human face by manually extracting each image pixel in many situations (e.g., face in shadow or glare environment, face with different emotions, etc) [25]. Therefore, DL can efficiently solve these problems by automatically learning the features from raw data and processing complex data, such as images, voice, and video.

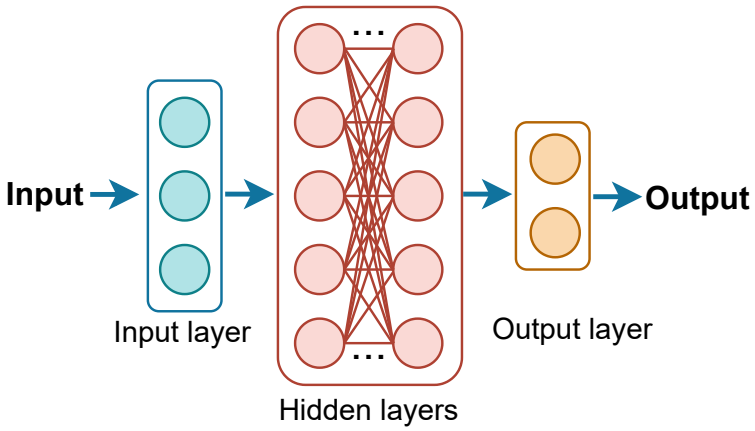



Fig. 1.3 Deep neural network architecture.

- *Ability to process big data*: In ML tasks, the more data resources that the algorithm learns, the more optimized and accurate performance it achieves. DL provides the ability to learn not only complex but also massive amounts of data, which is often limited in traditional ML. Instead of training large datasets on a single computing hardware (e.g., CPU and GPU), DL leverages parallel processing that utilizes the computing power of multiple computers for effective training. There are two types of parallel computing: model parallelism and data parallelism. Regarding the model, different layers of the DL model can be processed under multiple devices. Meanwhile, data parallelism allows a single DL model to learn different partitions of datasets in multiple devices.

- *Reusable*: Based on the effectiveness of the DL model in data generalization, the trained model can be reused for different purposes in several tasks. By using the pre-trained model built by experts, the costs for learning tasks can be significantly minimized with a few configurations. For instance, BERT, a famous pre-trained language model, can be applied to various domains (e.g., code analysis, medical-related documents analysis or social network analysis) without training from scratch for each task [27]. Furthermore, the trained DL model can be formed as a module where its components can be reused for different tasks within a complex system.

There are several types of DL models, including deep neural networks (DNN), convolutional neural networks (CNN), recurrent neural networks (RNN), deep autoencoders (DAE), and generative adversarial networks (GAN) [25], [28]. Although such DL models operate for different purposes, they basically consist of the same components: neuron layers, weights,

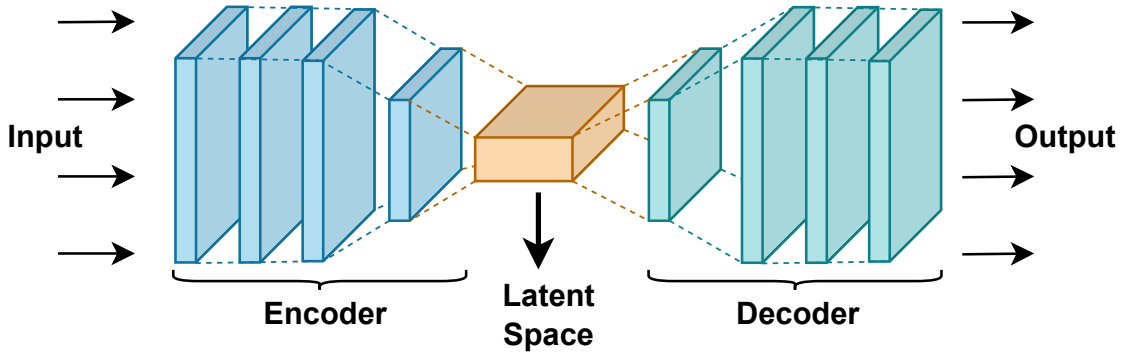



Fig. 1.4 Deep autoencoder architecture.

biases, activation functions, loss functions, and optimizers. Generally, the activation function calculates output based on the input from the neuron layer. The weight and bias of the neuron layer are then updated using the predefined loss function and optimizer to gain the optimal model.

### Deep Neural Network

The DNN is a typical DL model type, also known as the artificial neural network [25]. In particular, DNN is a feed-forward neural network characterised by multiple layers: an input layer, hidden layers, and an output layer, as described in Figure 1.3. Each linear layer consists of neurons that transform input data through weights, biases, and activation functions. This input is passed through multiple hidden layers to the output layer to produce the final result (e.g., classification label, predicted value), which is often called the forward process. In this process, the activation function at each layer provides non-linearity to the DNN, allowing it to learn complex patterns and features from the input data. Three widely adopted activation functions in DL models are Sigmoid, Tanh, and ReLU, each contributing to the network's ability to generalize intricate data relationships effectively [25] [26]. To optimize the output of a DNN during the learning process, backpropagation is employed to minimize the error between predicted results and actual labels. This key training task optimizes the weights and biases by leveraging gradient updates guided by an optimizer. Through this iterative process, the model continually improves its accuracy and performance. As a result, DNN is widely adopted in DL applications due to its effectiveness for advanced learning algorithms.

### Deep Autoencoder

The DAE is a type of DNN used primarily for unsupervised learning aimed at discovering efficient data representations, typically for dimensionality reduction or feature learning [25].

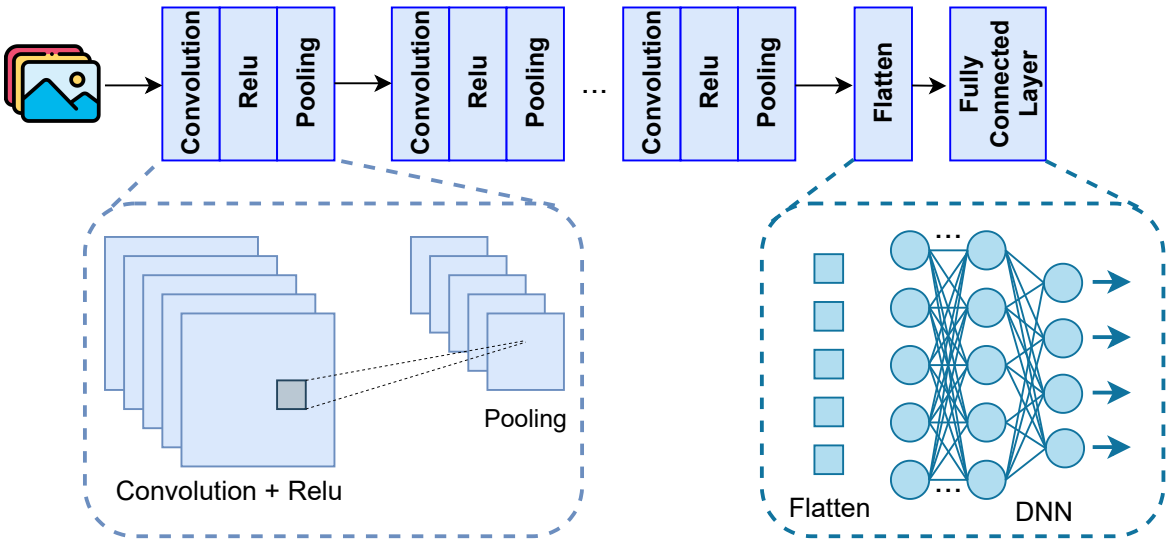



Fig. 1.5 Convolutional neural network architecture.

It leverages the "deep" nature of DL to improve the performance of the traditional autoencoder, which lacks robustness and accurate data reconstruction. As described in Figure 1.4, DAE contains two main parts: the encoder and the decoder. The encoder compresses the input data into a lower-dimensional latent space, capturing the most important features of the data. The decoder then reconstructs the original data from this compressed representation. By utilizing multiple neuron layers from DL in both the encoder and decoder, DAE can learn more complex and hierarchical features. In particular, DAE is highly effective for unsupervised anomaly detection, making them well-suited for NIDS to identify new cyberattacks.

### Convolutional Neural Network

The CNN is an advanced DNN mainly designed for handling visual data (e.g., image and video) [29]. Therefore, besides the well-known neural network, the architecture of CNN consists of two new layers: the convolutional layer and the pooling layer, as illustrated in Figure 1.5.

- *Convolutional layer*: specialises in feature extraction by using filters that slide over the input image, performing element-wise multiplications and summing the results [29]. This layer generates a feature map which contains the specific features obtained by the filters across different spatial locations within the input.

- *Relu activation function*: is employed to ensure the non-linearity of the CNN after extracting features from the convolutional layer. Specifically, it converts the negative

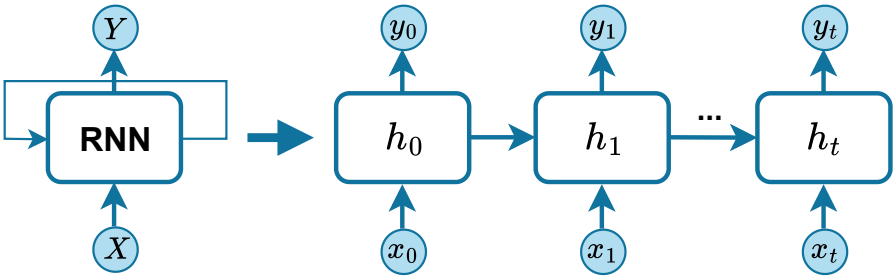



Fig. 1.6 Recurrent neural network architecture.

values to zero and maintains the positive ones during learning tasks, enabling CNN to train more efficiently.

- *Pooling layer*: aims to reduce the spatial dimensions of the feature map by utilizing the scanning window to process each value (e.g., pixel). Thus, it minimizes the parameters the network needs to learn, enabling more efficient computation and maintaining important features within less information.

After processing via the convolutional layer and pooling layer, CNN flattened the feature map into a vector to feed into fully connected layers, which is a neural network, to perform the classification tasks. Traditional ML algorithms and DNNs often struggle with image data because they require raw images to be converted into vectors before processing, leading to a loss of spatial information. In contrast, CNNs overcome those limitations by leveraging convolutional layers that automatically extract and learn complex features directly from the raw images [29]. This allows CNNs to outperform traditional models by preserving spatial relationships and capturing intricate patterns within the data. Regarding the NIDS, CNN is a potential technique for analyzing network data such as traffic flow and bytecode to identify cyberattacks.

### Recurrent Neural Network

The RNN is another type of DL model that enables sequential data processing (e.g., sensor signal, brain signal, weather data, audio data, etc.) [30]. Unlike the traditional feed-forward model (e.g., DNN and CNN), RNN leverages the previous information to improve the learning process of sequential data. As presented in Figure 1.6, the RNN is built upon directed cycle connections between nodes, enabling the retention of past information in the hidden state, which is then combined with new input data to inform the current step [30]. Particularly, the output of RNN in step $t-1$ is retained in hidden state $h_t$ and then used to improve the learning of new input at step $t$ [30]. Based on the architecture of RNN, several

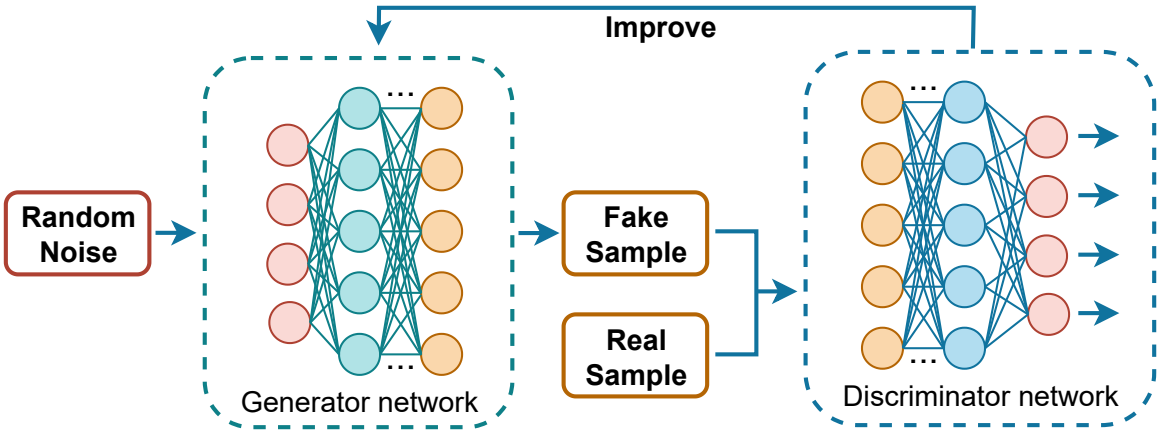



Fig. 1.7 Generative adversarial network architecture.

variants have been developed to solve the "gradient vanishing" challenge, including the long short-term memory (LSTM) and gradient recurrent unit (GRU) [30], [31]. As a result, RNN and its variants have shown the potential to analyze network data since network traffic is considered sequential.

#### Generative Adversarial Network

GAN is an advanced ML framework that employs the idea of adversarial learning to train the DL model [28]. As described in Figure 1.7, GAN consists of two neural networks, the generator and the discriminator network, aiming to generate new data with the same distribution as the training data. Specifically, the generator network takes the random noise as input and converts it into a synthetic sample, which is indistinguishable from the real sample [28]. The discriminator network then identifies the generated data and real data to accurately classify whether the input data is real or fake [28]. In this regard, the generator network tries to fool the discriminator by producing realistic data, while the discriminator tries to classify real and fake data. This adversarial learning strategy enables accurate synthetic data generation, thereby improving the learning of normal distribution in NIDS to detect abnormal activities in IoT networks.

### 1.2.2 Federated Learning

Federated learning (FL) is a decentralized ML method that enables multiple devices or entities to collaboratively train a shared model while keeping their data localized [32]. Unlike traditional methods that require centralized data collection, federated learning allows data to remain on individual devices, ensuring better data privacy and reducing communication

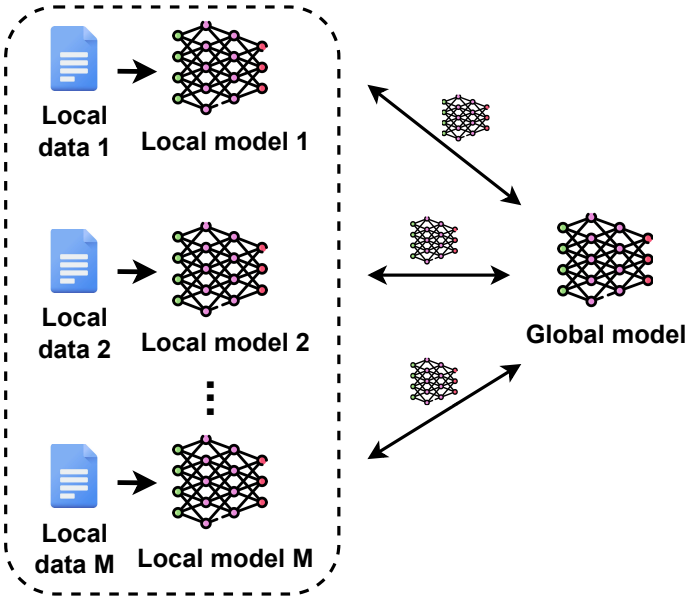



Fig. 1.8 Illustration of federated learning process.

overhead. As described in Figure 1.8, the FL system consists of a centralized FL server and $M$ FL users. In particular, each FL user $m \in \mathscr{M}$, $\mathscr{M} = \{1, \ldots, M\}$, that participate to FL system has its private dataset $D_m$. The FL process commences with the centralized server transmitting its initial global model $w_g$ to $M$ users. Upon receiving the global model, each user starts the training process by using their private dataset $D_m$ to produce the local gradient $g_m$ and update their local model $w_m$ [32]:

$$w_m^l \leftarrow w_g^l - \mu \cdot g_m^l, \tag{1.1}$$

where $l$ and $\mu$ are the learning round and learning rate, respectively. Subsequently, the users transmit their updated local model to the server for aggregation. During this stage, the server combines the received local models using the Federated Averaging (FedAvg) algorithm to update the global model effectively [32]:

$$w_g^{l+1} = \sum_{i=1}^{M} \frac{n_i}{n} \cdot w_i^l. \tag{1.2}$$

where $M$ is the total number of participating users, $n_i$ is the number of samples of user $i$, and $n$ is the total number of samples. The aggregated global model is subsequently sent back to the users for the next FL training round. This process continues either until the global model reaches convergence or until a specified number of training rounds has been completed. This weighted averaging ensures that client contributions reflect the size of their datasets, enabling

Table 1.1 Comparison of Different Enabling Cryptographic Methods for PPML

| **Methods** | **Interactivity** | **Encrypted Computation** | **Privacy Guarantee** | **Computation Overhead** | **Communication Overhead** |
|---|---|---|---|---|---|
| **SMPC** [21] | Non-Interactive | No | Strong | Medium | High |
| **DP** [34] | Interactive | No | Statistical | Low | Low |
| **HE** [33] | Interactive | Yes | Strong | High | High |

federated learning to harness distributed data efficiently. At the same time, it facilitates decentralized knowledge sharing, allowing participants to collaboratively improve the global model while preserving the privacy of their local information.

### 1.2.3 Homomorphic Encryption

Privacy-Preserving Machine Learning (PPML) leverages various cryptographic techniques, including Differential Privacy (DP), Secure Multi-Party Computation (SMPC), and Homomorphic Encryption (HE) [21]. Among these, HE stands out as a promising approach in which this cryptography method allows direct mathematical operation over encrypted data without decryption [33]. Unlike SMPC, which requires interactive communication overhead among multiple parties, HE operates efficiently in a non-interactive manner due to its ability in encrypted computation [21]. While DP protects privacy by adding noise to the data, this often leads to a trade-off between privacy and accuracy, potentially degrading model performance. In contrast, as a quantum-resistant scheme, HE guarantees both strong privacy and exact computations, preserving data integrity without any statistical distortions. Generally, Table 1.1 illustrates a comparison of current cryptographic methods regarding PPML applications.

Since first introduced, various HE schemes have been developed, including Brakerski-Gentry-Vaikuntanathan (BGV), Brakerski-Fan-Vercauteren (BFV), Fast Bootstrapping-Torus FHE (TFHE), and Cheon-Kim-Kim-Song (CKKS). However, due to the complexity in performing computation directly on HE-ciphertext, each scheme offers different trade-offs in terms of supported operations, computational precision, and efficiency, as illustrated in Table 1.2. As can be seen, in order to integrate HE with the neural network, we propose to use the Cheon-Kim-Kim-Song (CKKS) scheme, which enables approximate calculations on floating-point numbers [35]. Additionally, the CKKS allows for the encoding of multiple data segments into a single ciphertext, which is subsequently encrypted into a ciphertext, facilitating parallel computation on encrypted vector in a SIMD manner [35]. Since the CKKS

Table 1.2 Comparison of Different HE Schemes

| Scheme | Supported Operations | Supported Number | Precision | Multiplicative Depth | Efficiency |
|---|---|---|---|---|---|
| **CKKS** [35] | Addition & Multiplication | Floating-point | Approximate | Supports deep computation | High |
| **BFV** [36] | Addition & Multiplication | Integer | Exact Integer Arithmetic | Integer only | Moderate |
| **BGV** [37] | Addition & Multiplication | Integer | Exact Integer Arithmetic | Optimized for structured data | Moderate |
| **TFHE** [38] | Boolean Operations | Binary | High | Boolean logic | Low |

scheme is based on the ring learning with errors (RLWE), it requires the ring dimension $\mathscr{R}$ (a power-of-two integer) to ensure the security level and multiplicative depth [33]. Following that, the size of ciphertext (i.e., the maximum number of plaintexts it can contain) is denoted as $\mathscr{B}$, in which $\mathscr{B} = \mathscr{R}/2$. To be more specific, the CKKS scheme includes the key generation, encryption, decryption, and basic homomorphic operations as follows:

- $\mathsf{SKGen}(n)$: generate random secret key $\mathsf{sk}_n$ for user $n$.
- $\mathsf{PKGen}(\mathsf{sk}_n)$: create the public key $\mathsf{pk}_n$ for user $n$ based on the secret key $\mathsf{sk}_n$.
- $\mathsf{Enc}(\mathsf{pk}_n, c)$: encrypt a raw vector $\mathbf{c}$ into a ciphertext $\hat{\boldsymbol{c}}$ by using the public key $\mathsf{pk}_n$.
- $\mathsf{Dec}(\mathsf{sk}_n, \hat{\boldsymbol{c}})$: decrypt encrypted vector $\hat{\boldsymbol{c}}$ into its plain form $c$ by using the secret key $\mathsf{sk}_n$
- $\mathsf{Add}(\hat{\boldsymbol{c}}_\mathbf{1}, \hat{\boldsymbol{c}}_\mathbf{2})$: the addition between two ciphertexts $\hat{\boldsymbol{c}}_\mathbf{1}$ and $\hat{\boldsymbol{c}}_\mathbf{2}$, in which $\mathsf{Dec}(\mathsf{sk}_n, \mathsf{Add}(\hat{\boldsymbol{c}}_\mathbf{1}, \hat{\boldsymbol{c}}_\mathbf{2})) \approx \mathbf{c}_\mathbf{1} + \mathbf{c}_\mathbf{2}$.
- $\mathsf{Sub}(\hat{\boldsymbol{c}}_\mathbf{1}, \hat{\boldsymbol{c}}_\mathbf{2})$: the subtraction between two ciphertexts $\hat{\boldsymbol{c}}_\mathbf{1}$ and $\hat{\boldsymbol{c}}_\mathbf{2}$, in which $\mathsf{Dec}(\mathsf{sk}_n, \mathsf{Sub}(\hat{\boldsymbol{c}}_\mathbf{1}, \hat{\boldsymbol{c}}_\mathbf{2})) \approx \mathbf{c}_\mathbf{1} - \mathbf{c}_\mathbf{2}$.
- $\mathsf{Mult}(\hat{\boldsymbol{c}}_\mathbf{1}, \hat{\boldsymbol{c}}_\mathbf{2})$: the multiplication between two ciphertexts $\hat{\boldsymbol{c}}_\mathbf{1}$ and $\hat{\boldsymbol{c}}_\mathbf{2}$, in which $\mathsf{Dec}(\mathsf{sk}_n, \mathsf{Mult}(\hat{\boldsymbol{c}}_\mathbf{1}, \hat{\boldsymbol{c}}_\mathbf{2})) \approx \mathbf{c}_\mathbf{1} \times \mathbf{c}_\mathbf{2}$.

It is worth noting that the aforementioned homomorphic evaluations (i.e., addition, subtraction, and multiplication) perform element-wise operations between ciphertexts to produce

the encrypted outputs. However, the more homomorphic evaluations on ciphertext, the more noises are added, which causes the error in decryption [33]. To solve that problem, the bootstrapping mechanism denoted as $\mathsf{Bootstrap}(\hat{\boldsymbol{c}})$ can be applied to reduce the magnitude of noise in ciphertext $\hat{\boldsymbol{c}}$ by re-encrypting it [33]. This capability enables additional computation on ciphertext, making it suitable for deep learning tasks, including training and inference.

## 1.3 Literature Review and Contribution

### 1.3.1 Privacy-Aware Federated Learning for Intrusion Detection in IoV Networks

#### Literature Review

In the literature, several works have explored ML-based intrusion detection systems tailored for IoT and IoV networks. Particularly, ML and DL models serve as intelligent filters which analyze network traffic to effectively detect and classify incoming cyberattacks [39]. In [40], the authors built an IoT environment in their laboratory and produced a dataset called N-BaIoT, which contains network traffic from various benign IoT devices and malicious IoT botnets (e.g., Mirai and BASHLITE botnets). Then, a deep autoencoder (DAE) is developed and trained with the N-BaIoT dataset for anomaly detection. The simulation results indicated that the DAE can obtain 100% accuracy regarding True Positive Rate (TPR). Additionally, the authors in [41] also deployed a small-scale IoT system and conducted experiments to generate the ToN-IoT dataset. This dataset extends the capabilities of N-BaIoT by incorporating a broader range of attack scenarios, such as Man-in-the-Middle, Injection, Ransomware, and Scanning, while also providing telemetry data from IoT sensors under normal conditions. The authors then used the DNN and other traditional ML methods (e.g., Naive Bayes, Logistic Regression, Decision Tree) to evaluate the considered dataset. The experiment results showed that all ML models achieve the performance with Area Under the Curve (AUC) around 90%. Moreover, the authors in [42] proposed a novel generative model called Constrained Twin Variational Auto-Encoder (CTVAE), which can enhance representation learning in intrusion detection systems of IoT networks. By constructing a twin neural network to improve the latent space of the Variational Auto-Encoder (VAE), the reconstruction can be more separated due to different kinds of attacks. The simulation results clarify that the proposed approach can detect network attacks on IoT devices with accuracy from nearly 90% to 96% with the evaluation of real-world IoT datasets. Regarding IoV networks, the authors in [43] propose a Convolutional Long Short Term Memory Network (ConvLSTM) to detect anomalies in IoT sensors integrated CAVs. The simulation results show that the deep learning model

can detect various anomalies in sensor data with an F1-score of 97%. Moreover, in [44], the authors evaluate various deep learning techniques to detect attacks in vehicular network traffic, which achieve accuracy from 92% to nearly 99%. However, the aforementioned works primarily focus on centralized DL models for cyberattack detection, which rely heavily on accessing large datasets. These datasets are typically stored locally on user devices (e.g., IoT sensors, gateways, smartphones and vehicles), making them challenging to collect and aggregate. This reliance on a centralized framework not only increases privacy risks but also adds significant communication overhead, limiting the practicality and effectiveness of ML-based intrusion detection systems in IoT/IoV networks.

Regarding distributed processing for ML, numerous research works have explored the application of FL to cyberattack detection, particularly in environments with a vast number of devices, such as IoT ecosystems. In [45], the authors propose a collaborative framework that utilizes FL for intrusion detection in IoT networks. In this framework, DL models deployed on various IoT gateways utilize data collected from their respective subnetworks for local training. The locally trained models are then transmitted to a central server, where they are aggregated, enabling knowledge sharing and collaboration across diverse IoT infrastructures. Experiment results using Deep Belief Network (DBN) and Deep Autoencoder (DAE) show the accuracy of detection from 93% to 98%. Besides, the authors in [46] employ both Transfer Learning (TL) and FL to create the advanced DL model, which can learn different features from different IoT networks. In the considerer framework, each IoT network may have different extracted features and labels of the input data. Therefore, the integration of TL and FL is used to transfer the knowledge between various IoT subnetworks. The proposed framework can help the learning models transfer the knowledge of labelled and unlabeled data to classify different attacks. The proposed Federated Transfer Learning (FTL) shows that the FTL can detect network attacks on IoT devices with accuracy from 85% to 99%. Moreover, the authors in [47] also showed the potential of FL for IDS in vehicular networks. The authors designed a blockchain-based FL where the Roadside Units (RSUs) can effectively collect traffic data from the vehicle and subsequently use it to train the considered local DL model. The simulation results showed that with different numbers of vehicles and portions of local data, the proposed framework can achieve consistent results of approximately 95% regarding accuracy, precision, and recall.

Despite the advantage of FL in IDSs, there are still some major challenges when deploying it in practical IoV networks. Specifically, in FL-based IDS in IoV networks, vehicles or RSUs often serve as workers to store and process all the learning tasks (e.g., training and classification) [48]. However, in practice, both RSUs and vehicles usually have limited computing and storage resources, and thus, storing and processing learning tasks at RSUs

and vehicles are ineffective. It is important to note that in conventional FL processes, a delay from one computing node can cause a delay for the whole system [49]. Therefore, several works propose solutions to upload data from RSUs and vehicle users (VUs) to powerful servers (e.g., centralized servers) for processing [50] [51]. The authors in [50] propose a hybrid FL framework (HybridFL) in which users can choose to offload their full data to the server. The experiment results reveal that HybridFL can slightly improve the accuracy of the DL model with 1% of data-uploading clients. In addition, the work in [51] design the offloading framework, which assists edge FL in wireless networks. In the considered framework, the client (e.g., mobile users) can freely choose the amount of data they want to share with the server to employ such high computing power of the server. These approaches can be very effective in deploying ML algorithms as all the data is collected and processed at the centralized servers. However, it also raises a serious concern regarding the data privacy of VUs, as all the data is now stored and processed externally [52].

### Contributions

To overcome the above challenges, we propose a novel privacy-preserving FL framework for intrusion detection in IoV networks. The proposed framework can effectively protect VUs' privacy and detect cyberattacks, given VUs' limited computational resources. Our main contributions can be summarized as follows.

- We design a novel framework for efficient computational resources in FL. Based on the current computing and storage resource capabilities, VUs can decide the amount of data they need to upload to the server for processing. Therefore, the users can locally process the data with the resource-constrained IoV devices.
- We then introduce the approach to effectively protect users' privacy during the offloading process. Specifically, the offloading data will be encrypted by employing Homomorphic Encryption (HE) before being uploaded to the server.
- While this encryption method enhances data privacy, it presents significant challenges for the centralized server, which is tasked with training on the encrypted data offloaded from the VUs. To tackle this issue, we develop a robust training algorithm leveraging the Single Instruction Multiple Data (SIMD) and the bootstrapping capabilities of the underlying HE scheme. This enables direct computation on the quantum-secure encrypted ciphertexts without the need for decryption. This approach not only maintains the confidentiality of data during the offloading process from VUs to the centralized server but also boosts the efficiency of using FL for IDSs within IoV networks.

- We carry out extensive simulation results to evaluate the proposed framework on real-world IoT datasets. In particular, our proposed framework achieves not only a high accuracy (approximately 91%) in detecting attacks but also exhibits great performance, closely approaching the benchmark without using encryption (with a gap of less than 0.8%)

### 1.3.2 Privacy-Preserving Machine Learning for Cyberattack Detection in Blockchain-based IoT Networks

#### Literature Review

As described in Section 1.1, with the increasing reliance of IoT systems on blockchain technology for security and data integrity, cyberattacks targeting blockchain networks pose significant risks to the stability and functionality of IoT infrastructures. To protect the blockchain networks, numerous studies have explored the application of intrusion detection methods to detect and prevent specific blockchain network attacks. The authors in [53] analyzed the flooding of transactions (FoT) attack on the Moreno blockchain in terms of the network capacity, block size, etc., to evaluate the attacker's benefits. The simulation indicates that the attacker can potentially retrieve the users' data in nearly 41% of all transactions. In [54], besides analyzing the cyberattacks (i.e., DDoS, brute passwords) on the Ethereum nodes, the authors also implemented the detection of cyberattacks based on scanning techniques. The authors showed that they can detect cyberattacks based on CPU consumption and high memory on various devices (e.g., Macbook, Mobile devices, and Raspberry Pi). However, the above approaches can only detect several specific types of attacks and often detect threats only after significant damage has already been incurred.

Unlike the conventional methods, ML enhances the cyberattack detection performance by allowing the model to learn from the distribution patterns of both normal and attack data, enabling it to identify multiple types of attacks [18]. In addition, the ML model is capable of detecting new, previously unreported attacks, making it particularly effective for blockchain environments that frequently encounter novel cyber threats [19], [55]. In [56], the authors proposed a federated learning approach to improve the IDS in IoT-based blockchain applications for Metaverse. Specifically, a semi-supervised learning model is designed to detect network attacks in IoT environments. The evaluation of multiple IoT attack detection datasets shows that the considered learning model can detect attacks with accuracy from 82% to nearly 98%. In addition, the authors in [57] proposed a deep learning (DL) model which combines the contractive sparse autoencoder (CSAE) and Long Short-Term Memory (LSTM) to detect cyberattacks on IoT-based blockchain systems. The simulation results on

two well-known ToN-IoT and Edge-IIoT datasets show that the considered DL model can achieve accuracy from 90% to nearly 99% in different scenarios. However, the application of ML to detect cyberattacks in blockchain network traffic is still in its infancy since it has only been explored in a few studies. In [58], the authors analyzed the Bitcoin traffic data and experimented with DoS and Eclipse attacks to collect network attack data. The authors then designed an autoencoder (AE) to detect the considered attacks on the Bitcoin traffic in which the accuracy of the considered AE reached nearly 99%. Besides, the work in [59] utilized the Recurrent Neural Network (RNN) to detect various DDoS attacks on Ethereum networks. The authors designed a simulated Ethereum network and collected the normal and attack behaviours in terms of network records. The results showed that the considered approach can detect attacks with 99% accuracy. Regarding the blockchain-based IoT networks, the authors in [60] developed a private blockchain-based IoT for supply chain management and experimented with various network attacks to collect the dataset. After that, the authors proposed to integrate DAE with DNN, forming an effective semi-supervised DL model for anomaly detection. The simulation results indicated that the proposed DL model can detect anomalies with an accuracy of around 96%. Moreover, the authors in [61] designed a private Ethereum network to process data transmission from IoT devices. The benign and attack traffic collected from IoT devices and Ethereum nodes was used to train a Deep Belief Network (DBN), achieving 98% accuracy in detecting cyberattacks.

Nevertheless, as described in Section 1.1, in blockchain-based IoT systems, applying ML for cyberattack detection raises significant privacy concerns due to reliance on third-party services like cloud providers, which require transferring sensitive user data for processing. While distributed techniques such as federated learning reduce dependency on centralized training, they are limited by the hardware constraints of IoT gateways, which must balance the demanding tasks of blockchain mining and FL local model training. Therefore, privacy-preserving ML using HE emerges as an innovative solution to ensure data confidentiality throughout the computation process. Although differential privacy is a widely recognized technique in privacy-preserving ML, especially when integrated with FL, it faces limitations in ensuring user data privacy as this remains an essential concern in blockchain-based IoT systems.

To preserve the privacy of users, various works have successfully integrated HE into ML models. In [62], the authors proposed a neural network integrated with HE utilizing non-linear activation functions. By evaluating the considered neural network on the MNIST dataset, the authors showed that the HE-encrypted neural network can achieve an accuracy of 99% and high throughput in the encryption and decryption process. Additionally, the authors in [63] introduced Doren, a DL-based HE method that allows the processing of a

Deep Convolutional Neural Network (CNN) over HE-encrypted data. By employing the SIMD packing and bootstrapping techniques, the authors enable the HE computation on multiple famous CNN models (i.e., VGG7, ResNet20). The simulation results indicated that the proposed approach could achieve accuracy from 73.85% to 92.32% within multiple HE schemes and DL models. Besides, in [64], the authors proposed a CNN-integrated HE method to recognize human activities in a privacy-preserved manner. The authors also based on the SIMD packing method to design effective HE-encrypted matrix multiplication, forming an effective encrypted CNN with high throughput during inference. The results showed that the proposed approach could have a high throughput of 0.4 to 0.8 seconds per sample and accuracy from 86% to nearly 89% within multiple datasets and CNN configurations. However, it is worth noting that while these studies successfully handle the inference process in ML-integrated HE, they do not address the complexities of the training process, which is arguably the most critical phase in machine learning.

In [65], the authors proposed CryptoDL, which considered both training and inference on the HE-encrypted data. However, the proposed CryptoDL requires frequent communication between ML owners and users to refresh the encrypted parameters, preventing the overload of HE noise during the training process. Alternatively, the proposed approach costs massive training time, resulting in around 1,456.7 seconds to train over one iteration with a 5-layer neural network. The authors in [66] also considered the training task of neural network-integrated HE with a multi-threading approach. The results showed that the training task of a 3-layer neural network requires intensive training time. Specifically, it takes over 9 hours with a single thread and 40 minutes with 30 threads to process a mini-batch of 60 samples. In [67], the authors developed a training algorithm for an encrypted neural network that achieves nearly 88% accuracy in recognizing human activities, yet they did not take the training time into consideration.

As discussed above, relying on ML to enhance blockchain-based IoT security raises notable privacy concerns. Given HE's proven effectiveness in safeguarding privacy when combined with ML, training DL models on HE data presents two significant challenges, i.e., computational overhead and limited efficient operations. Generally, HE enables computations on encrypted data without needing to decrypt it, which is crucial for privacy-preserving ML [21]. However, it comes at the cost of extensive computational overhead, as operations on ciphertext require much more resources than on plaintext. Therefore, the extensive computational requirement for training tasks of ML-integrated HE leads to prohibitively long training time, making it challenging to scale DL models to large datasets when preserving users' privacy by HE. Moreover, since HE schemes only efficiently support a limited set of mathematical operations (i.e., addition and multiplication), effectively implementing ML

training tasks for HE-encrypted data (matrix multiplication, back-propagation, non-linear functions, etc.) is particularly challenging.

### Contributions

To address all the above challenges, we propose a novel privacy-preserving cyberattack detection framework for blockchain-based IoT systems. In the proposed system, AI-based smart cyberattack detection modules are deployed at the mining nodes in the blockchain network to detect attacks on the mining nodes in a real-time manner. To improve the efficiency in detecting attacks in real-time (i.e., with high accuracy and low delay), the training process is performed in advance at a cloud service provider (CSP). In particular, the mining nodes send their training data to the CSP for a comprehensive training process. To protect data privacy, before sending the training data to the CSP, the mining nodes encrypt their data using the HE technique. This technique allows the CSP to perform global model training on encrypted data without the need to decrypt it, thereby protecting data privacy. To address the problem of handling a huge amount of encrypted data at the CSP, we first develop an innovative packing algorithm to pack the data in an SIMD manner, thereby effectively enabling the training process for HE-encrypted data. After that, we design an innovative training algorithm for the deep neural network with HE-encrypted data based on our proposed packing methods. While applying HE to ML/DL models can effectively enhance data privacy, it presents significant challenges for the CSP regarding computation time during the training task. To tackle this, we propose a privacy-preserving distributed learning for HE-encrypted data. Our proposed approach allows the learning model to be trained in parallel across multiple workers, thereby improving computation time. Once the training process is completed, the CSP will share the trained model with the mining nodes for real-time detection. Our main contributions are summarized as follows:

- We develop innovative packing methods for 1D vector and 2D matrix, which allow the encrypted matrix multiplication in an SIMD manner. Following that, we design a robust training algorithm for encrypted data based on the proposed packing algorithm mentioned above.
- We propose a privacy-preserving distributed learning algorithm that significantly optimizes the training time over the encrypted data. By leveraging the FedAvg algorithm, computing resources from multiple parties can be utilized to form an effective learning approach. The experiments show that the proposed learning algorithm reduces training time significantly as more workers participate.

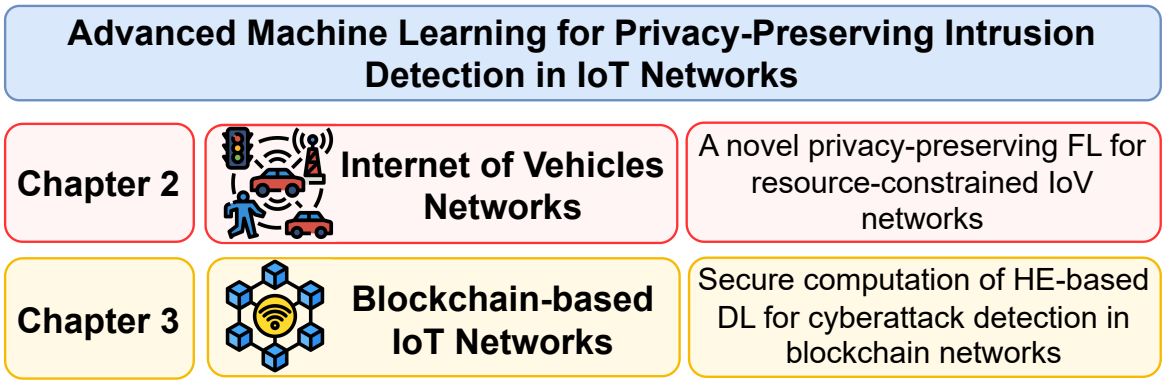



Fig. 1.9 The structure of the thesis.

- We conduct comprehensive evaluations on the detection performance with the real-world blockchain network attack dataset. The results show that our approach not only delivers highly accurate cyberattack detection for both encrypted and non-encrypted inference but also achieves consistency in performance comparable to the non-encrypted baseline method. This validates the robustness and reliability of our solution in maintaining privacy without compromising detection accuracy.

- We perform real experiments to evaluate our proposed framework across various consensus mechanisms and hardware configurations. The results indicate that our framework is highly adaptable to practical applications, delivering impressive efficiency in resource utilization, low latency, and high throughput. These findings underscore the framework's potential for seamless integration into real-world systems, meeting the demands of modern decentralized environments.

## 1.4 Thesis Organization

As illustrated in Figure 1.9, this thesis significantly contributes to two distinct IoT systems. Specifically, chapter 2 focuses on advancements in FL and HE for IDS in IoV networks, while chapter 3 provides a deeper exploration of time-efficient HE-based DL computations for cyberattack detection in blockchain-based IoT networks. The structure of this thesis is outlined as follows.

- Chapter 2: This chapter presents our study that enables homomorphic encryption in federated learning (FL) to provide a privacy-preserving approach for IDS in resource-constrained IoV networks. Specifically, Section 2.1 describes our system model. Then, Section 2.2 introduces our proposed DNN for encrypted data. Based on that, our pro-

posed privacy-preserving FL is described in Section 2.3. In Section 2.4, performance evaluations are discussed. Finally, Section 2.5 provides the conclusion of this work.

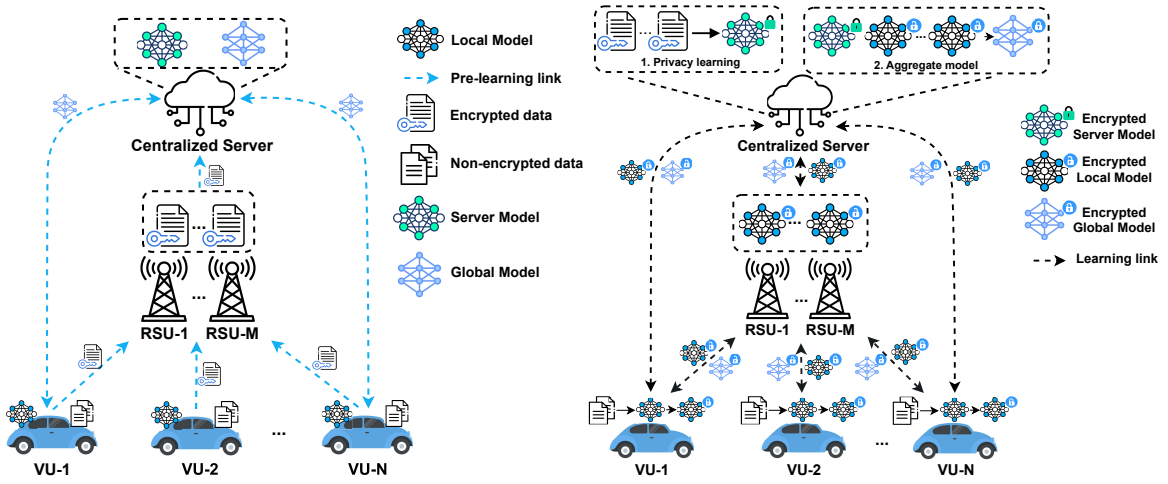
- Chapter 3: This chapter introduces a novel privacy-preserving cyberattack detection framework for blockchain-based Internet-of-Things (IoT) systems. Particularly, Section 3.1 demonstrates a brief overview of blockchain-based IoT systems and the system model. In Section 3.2, the proposed training algorithm of DNN for HE-encrypted data and distributed cloud-native learning are presented. Then, simulation results and real experiment results are discussed in Section 3.3. Finally, we conclude our work in Section 3.4.

- Chapter 4: This chapter summarizes the conclusions and outlines potential directions for future research.

# Chapter 2

# Homomorphic Encryption-Enabled Federated Learning for Privacy-Preserving Intrusion Detection in Resource-Constrained IoV Networks

This chapter aims to develop an innovative approach to ensure both security and privacy for IoV networks, which is an emerging IoT-based application. Particularly, we aim to propose a novel framework to address the data privacy issue for Federated Learning (FL)-based Intrusion Detection Systems (IDSs) in Internet-of-Vehicles (IoVs) with limited computational resources. In particular, in conventional FL systems, it is usually assumed that the computing nodes have sufficient computational resources to process the training tasks. However, in practical IoV systems, vehicles usually have limited computational resources to process intensive training tasks, compromising the effectiveness of deploying FL in IDSs. While offloading data from vehicles to the cloud can mitigate this issue, it introduces significant privacy concerns for vehicle users (VUs). To resolve this issue, we first propose a highly-effective framework using homomorphic encryption to secure data that requires offloading to a centralized server for processing. Furthermore, we develop an effective training algorithm tailored to handle the challenges of FL-based systems with encrypted data. This algorithm allows the centralized server to directly compute on quantum-secure encrypted ciphertexts without needing decryption. This approach not only safeguards data privacy during the offloading process from VUs to the centralized server but also enhances the efficiency of utilizing FL for IDSs in IoV systems. Our simulation results show that our proposed approach can achieve a performance that is as close to that of the solution without encryption, with a gap of less than 0.8%.




(a) Pre-learning process (b) Privacy-preserving learning process

Fig. 2.1 The proposed privacy-preserving intrusion detection framework including pre-learning and privacy-preserving learning.

## 2.1 System Model

The proposed system model is illustrated in Fig. 2.1. The system consists of a centralized server (CS), $M$ RSUs and $N$ VUs. Initially, the VUs enter the pre-learning phase by assessing their computational resources and determining the optimal amount of data that can be processed locally. The rest of the data will be offloaded to the centralized server for processing. However, before offloading the data to the centralized servers, VUs will generate HE key pairs and use them to encrypt uploading data. The encrypted data will then be offloaded to the centralized server via RSUs, as illustrated in Fig. 2.1(a). Upon receiving the offloaded encrypted data, the centralized server will compile it into an encrypted dataset. Using this dataset, two learning models will be developed: the server model and the global model, each serving distinct purposes. The server model will be utilized to train the encrypted dataset within the centralized server, whereas the global model will be distributed to the VUs for local training. Once the global model is sent to the VUs, the privacy-preserving learning process will commence.

The privacy-preserving learning process will be divided into different learning periods. During each learning period, each VU will use the global model to train on its local data. After completing the training, the VU will encrypt its trained model before sending it to the centralized server. Concurrently, the centralized server will train its encrypted data using our proposed CKKS scheme, detailed in Section 2.2. Upon receiving all the encrypted trained models from the VUs, the centralized server will aggregate them to create a new global model (the aggregation method is detailed in Section 2.3). This updated global model is then

sent back to the VUs, and the next learning period begins. This process repeats until the global model converges or until a predefined number of learning periods has been completed.

## 2.2 The Deep Neural Network for Encrypted Data

To integrate HE with deep neural networks, we propose to use the Cheon-Kim-Kim-Song (CKKS) scheme. The reason is that it allows the encryption and calculation of real numbers, which is suitable for deep learning [35].

As described in Section 1.2.3, HE schemes require a ring dimension $R$, which maintains the security level, multiplication depth, and noise level [33], thereby allowing accurate computations over encrypted data. Following that, to design a deep neural network for encrypted data, we employ the single instruction multiple data (SIMD) from the CKKS scheme, which packs multiple plaintexts into a single ciphertext. The size of ciphertext is denoted as $B$, where $B = R/2$. Alternatively, the CKKS can encode and encrypt a square matrix of size at most $\mu \times \mu$, where $\mu = \lfloor \sqrt{B} \rfloor$ by initially flattening it into a vector. This thus enables element-wise operation on the plaintext slots concurrently. For clarity, Fig. 2.2 describes the implementation of the weight matrix encryption method. Let $\phi_i$ denote the parameter of $i$ linear layer which $\phi_i = (W_i, b_i)$. The weight matrix $W_i^{u \times v}$ with $u$ and $v$ as the input and output dimensions of the layer is applied to the encoding process:

$$Encode(W_i) = Flatten\big(Pad(W_i, 0, \mu)\big), \tag{2.1}$$

where the matrix $W_i^{u \times v}$ is first zero-padded to $W_i^{\mu \times \mu}$ with the size of ($\mu$, $\mu$) to fit within $B$ size. The weight matrix is then flattened to form an encoded vector. After that, this encoded vector is padded to ensure its length equals half of the ring dimension [33]. After that, the encoded vector is encrypted using the encryption function of CKKS described in Section 1.2.3, which can be defined by:

$$\hat{W}_i = \mathsf{Enc}(s^{pk}, Encode(W_i)), \tag{2.2}$$

where $s^{pk}$ is the public key generated by the users. As a result, $\hat{W}_i$ is the encrypted weight of layer $i$, which can be used to operate with encrypted training data. Therefore, the output of the forward propagation over the $i$-th layer can be calculated as:

$$\hat{x}_{i+1} = \hat{\sigma}\big(\mathsf{Add}(\mathsf{Mult}(\hat{x}_i, \hat{W}_i), \hat{b}_i)\big), \tag{2.3}$$

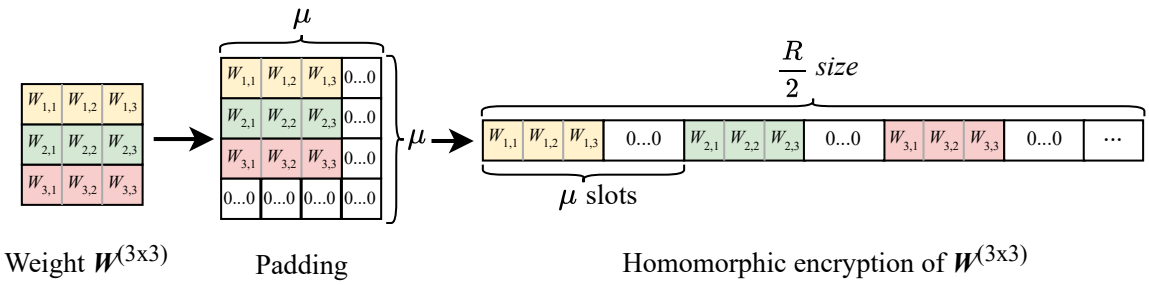



Fig. 2.2 Encrypt process of a 3x3 matrix $\boldsymbol{W}$.

where $\hat{W}_i$ and $\hat{b}_i$ are the encrypted weight and bias at layer $i$. Particularly, $\hat{\sigma}$ illustrates the polynomial approximation of the activation function $\sigma$ using the Chebyshev polynomial [68]. In the considered deep neural network, the *Swish* (SiLU) activation function is chosen due to its advantage in solving the "dying ReLU" problems [69]. Subsequently, the encrypted output vectors consist of the distribution of the classes for classification tasks. It is noted that the *Softmax* function is not applied in this work due to the exponential and inverse functions contained, which are non-homomorphic [67].

In the backpropagation process, we apply the Stochastic Gradient Descent (SGD) for mini-batch regarding the optimization. After calculating the encrypted gradients for each layer, the SGD update of the encrypted weight can be formulated as:

$$\hat{W}_i \leftarrow \mathsf{BootStrap}\Big(\mathsf{Sub}\Big(\hat{W}_i, \mathsf{Mult}\big(\eta, \frac{\partial \hat{L}}{\partial \hat{W}_i}\big)\Big)\Big). \tag{2.4}$$

where $\frac{\partial \hat{L}}{\partial \hat{W}_i}$ is the calculated encrypted gradient of $\hat{W}_i$, which is computed via the derivative of encrypted loss function $\hat{L}$. After the SGD update, the encrypted weight $\hat{W}_i$ is applied to the BootStrap method, which renews the ciphertext, allowing additional computation on $\hat{W}_i$ and reduces the magnitude of accumulated noise [33]. Generally, the conversion of the neural network (e.g., weights and non-linear activation functions) is illustrated in Fig. 2.3.

## 2.3 The Proposed FL Implementation

In the pre-learning phase, each VU-$n$ evaluates its computing resources and chooses $p_n$% of data to offload. After that, they generate a key pair, including a secret key $s_n^{sk}$ and a public key $s_n^{pk}$. In particular, the VU-$n$ divide its collected dataset $\mathscr{D}_n$ into the local dataset $\mathscr{DR}_n$ and offloaded dataset $\mathscr{DS}_n$ based on effective computing resources of the vehicles. The $\mathscr{DS}_n$ is then encrypted to $\hat{\mathscr{DS}}_n$ to protect the user data. The encrypted data is sent to the CS and

**Algorithm 1** Proposed Privacy-Preserving FL Framework

**for** $\forall n \in N$ **do**
  Calculate $p_n\%$ for offloading
  Generate a secret key and a public key: $s_n^{sk} = SKGen(n)$ and $s_n^{pk} = PKGen(s_n^{sk})$
  Split the dataset $\mathscr{D}_n$ into $\mathscr{DR}_n$ and $\mathscr{DS}_n$ where $\mathscr{DS}_n = \mathscr{D}_n \times p_n$ and $\mathscr{DR}_n = \mathscr{D}_n - \mathscr{DS}_n$
  Generate the encrypted data $\hat{\mathscr{DS}}_n = Enc(s_n^{pk}, \mathscr{DS}_n)$
  Send encrypted data $\hat{\mathscr{DS}}_n$ to the CS
**end for**
CS combines the received encrypted data $\hat{\mathscr{DS}}_n$ into $\hat{\mathscr{DS}}$
CS initializes the $\mathscr{M}_g$ and $\mathscr{M}_s = \mathscr{M}_g$
Transmit $\mathscr{M}_n$ to $N$ VUs which $\mathscr{M}_n = \mathscr{M}_g$
Generate encrypted model $\hat{\mathscr{M}}_s$ and $\hat{\mathscr{M}}_g$ via $s_n^{pk}$ where $\phi_n = Dec(\hat{\phi}_n)$
**while** $\tau \leq T_{\max}$ or training process does not converge **do**
  $\hat{\mathscr{M}}_s$ learns the encrypted data $\hat{\mathscr{DS}}$
  $\hat{\mathscr{M}}_s$ produces encrypted parameters $\hat{\phi}_s^{\tau}$
  **for** $n \in N$ **do**
    $\mathscr{M}_n$ learns the local data $\mathscr{DR}_n$
    Calculate local parameters $\phi_{\tau}^n$
    Encrypt local parameters $\hat{\phi}_{\tau}^n = Enc(s_n^{pk}, \phi_{\tau}^n)$
    Send local encrypted parameters to the CS.
  **end for**
  The CS calculates and produces the encrypted global model $\hat{\phi}_g^{(\tau+1)}$.
  Send the updated global model $\hat{\phi}_g^{(\tau+1)}$ back to $N$ VUs
  **for** $\forall n \in N$ **do**
    Decrypt the model $\phi_g^{(\tau+1)} = Dec(s_n^{sk}, \hat{\phi}_g^{(\tau+1)})$
  **end for**
**end while**
Predict $\hat{Y}_n$ based on the encrypted training data $\hat{X}_n$ at each VU-n and optimal global model $\phi^*$.

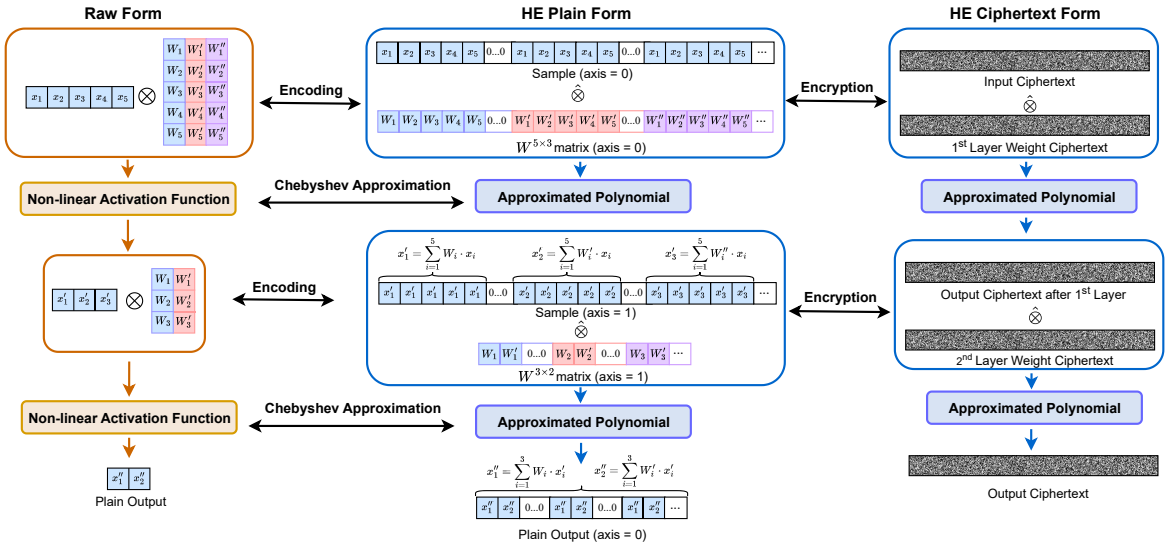



Fig. 2.3 The general flow of converting the weight and activation functions of neural network to HE dimension.

combined to form an encrypted dataset $\hat{\mathscr{DS}}$. After that, the CS initializes the non-encrypt global model $\mathscr{M}_g$ and non-encrypt server model $\mathscr{M}_s$, then distributes $\mathscr{M}_g$ to each VU for local training. Subsequently, the public key is used to initialize the encrypted learning model $\hat{\mathscr{M}}_s$ and encrypted global model $\hat{\mathscr{M}}_g$ on the server.

Regarding the privacy-preserving learning phase, we consider $T$ learning rounds. At each round $\tau$, privacy is maintained by the non-encrypted training from the local learning model of VU-$n$ and encrypted training from the privacy-preserving learning model. After finishing the $\tau^{\text{th}}$ local training round, VU-$n$ encrypts the trained parameters $\hat{\phi}_\tau^n$ and sends them to the CS. The CS retrieves the encrypted parameters from $\hat{\mathscr{M}}_s$ along with the local encrypted parameters and aggregates by the FedAvg algorithm for encrypted data, which can be defined by:

$$\hat{\phi}_g^{(\tau+1)} = Mul\left(\frac{1}{N+1}, Add\left(\sum_{n=1}^{N} \hat{\phi}_n^\tau, \hat{\phi}_s^\tau\right)\right). \tag{2.5}$$

The global parameters $\hat{\phi}_g^{(\tau+1)}$ are then updated to the encrypted global model $\hat{\mathscr{M}}_g$ and sent back to the VUs, which is then decrypted by the VUs for the next learning round. The learning process continues until the global model converges and obtains the optimized parameters.

In summary, the learning process of the privacy-preserving intrusion detection framework for IoV is described in Algorithm 1.

Table 2.1 The distribution of classes of the dataset

| Class | Number of samples |
|---|---|
| Normal | 5,320 |
| DDoS | 5,472 |
| MitM | 4,000 |
| Injection | 5,589 |
| Malware | 5,504 |
| Reconnaissance | 5,515 |
| Total | 31,400 |

Table 2.2 Parameters configuration of the neural network

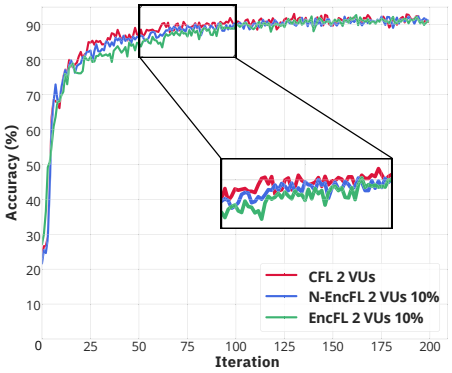
| Layer | Number of Neurons | Activation Function |
|---|---|---|
| Input | 32 | None |
| Hidden Layer 1 | 16 | Chebyshev SiLU |
| Hidden Layer 2 | 16 | Chebyshev SiLU |
| Output | 6 | Chebyshev SiLU |

## 2.4 Performance Evaluation

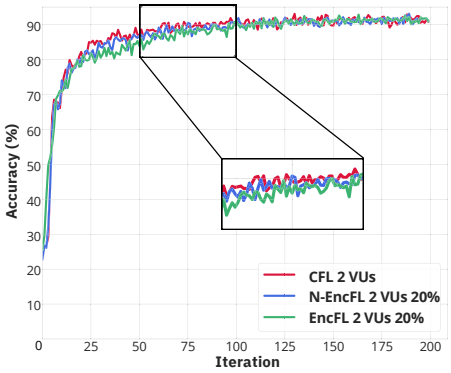
### 2.4.1 Simulation Setup

In this section, the proposed privacy-preserving model is validated on the real-world dataset of network traffic attacks on IoT devices, named Edge-IIoT dataset [70]. As shown in [70], [71], [72], the Edge-IIoT dataset covers various potential threats to IoT components (sensors, gateways and operation systems), which are commonly embedded to vehicles and RSUs in IoV systems. It includes 20 million raw normal traffic and attack traffic collected from 13 IoT devices. Attacks can be grouped into the five most common types, including Distributed Denial of Service (DDoS), Injection, Man-in-the-Middle (MitM), Malware, and Reconnaissance. After applying downsample and oversample to overcome the imbalance, the dataset includes 31,400 samples with details presented in Table 2.1. Subsequently, the dataset is divided into training and testing sets (80%-20%). The training and testing sets are then nominalized and scaled within the range of (0,1). Regarding the neural network, we design a fully connected network consisting of an input layer, 2 hidden layers, and an output layer. The respective layers contain 32, 16, 16, and 6 neurons. Apart from the input layer, each layer is attached to the SiLU activation function, as shown in Table 2.2.

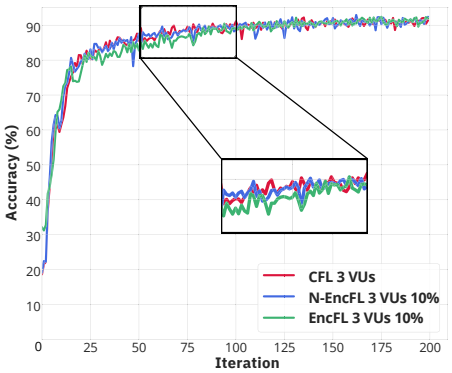
During the experimental evaluation, we assume that the collected dataset of each VU has the same class distribution. Similar to [51], we consider the approach to offload partial data to the server as the benchmark for our proposed framework. Nevertheless, it is noted



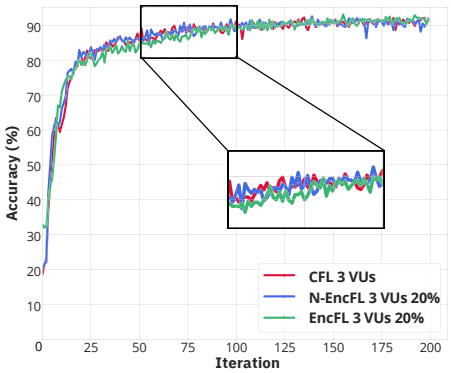

(a) 2 VUs with 10% offload data




(b) 2 VUs with 20% offload data




(c) 3 VUs with 10% offload data




(d) 3 VUs with 20% offload data

Fig. 2.4 Convergence of privacy-preserving learning with different number of VUs. Regardless of the amount of offloaded encrypted data, the proposed EncFL consistently achieves accuracy comparable to scenarios without decryption.

that in such a benchmark, the FL framework does not consider the privacy of the VUs. In this scenario, following [51], [67], we train the non-encrypted data using the non-encrypted model at both CS and VUs. Consequently, the trained global model is employed to evaluate the accuracy of our proposed framework and other benchmarks [67]. In our experiment setup, the proposed framework consists of 2 VUs and 3 VUs, which can send 10% and 20% of their local data.

### 2.4.2 Evaluation Metrics

To evaluate the performance of the detection model, the confusion matrix is utilized, which is suitable for a machine learning-based classification system [73]. We denote TP, TN, FP, and

FN as “True Positive”, “True Negative”, “False Positive”, and “False Negative”. Assuming the system consists of $C$ classes, which include normal and attack traffic, the accuracy can be calculated as:

$$Accuracy = \frac{1}{C}\sum_{c=1}^{C}\frac{TP_c + TN_c}{TP_c + TN_c + FP_c + FN_c}. \tag{2.6}$$

The macro-average precision and recall are utilized in this term. Given $K$ as the number of classes in the system, the macro-average precision is:

$$Precision = \frac{1}{K}\sum_{k=1}^{K}\frac{TP_k}{TP_k + FP_k}. \tag{2.7}$$

The macro-average recall is calculated as follows:

$$Recall = \frac{1}{K}\sum_{k=1}^{K}\frac{TP_k}{TP_k + FN_k}. \tag{2.8}$$

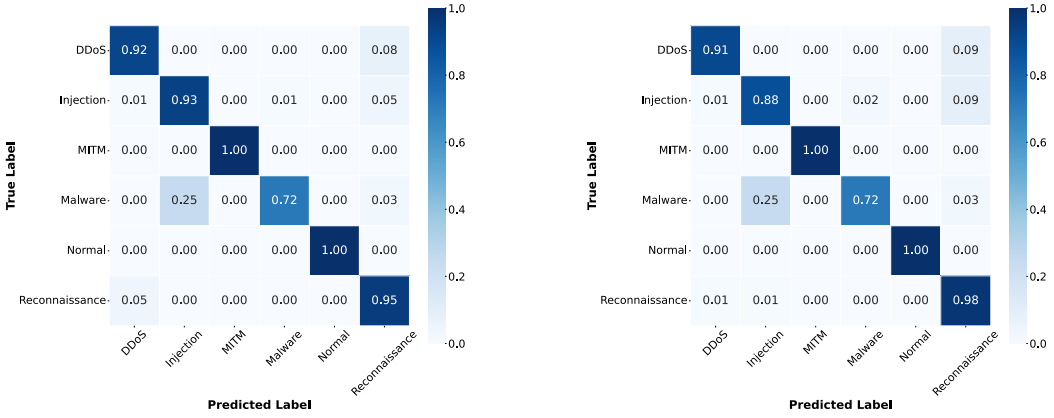
### 2.4.3 Simulation Results

#### Convergence Analysis

Fig. 2.4 illustrates the convergence of learning processes from three approaches, including conventional FL (CFL) [32], FL with non-encrypted offloaded data (N-EncFL) [51] and the proposed privacy-preserving learning (EncFL). As observed in Fig. 2.4(a) and Fig. 2.4(b) with 2 VUs, the CFL converges after 70 iterations, while the N-EncFL and EncFL require nearly 100 iterations to reach the convergence. Specifically, due to the different amounts of data handled by the VU, the CFL demonstrates a slightly better convergence rate compared to other approaches. Despite the trivial difference in convergence, the accuracy during the learning process of the three approaches remains nearly identical, stabilizing at approximately 92%. Additionally, Fig. 2.4(c) and Fig. 2.4(d) describe the convergences in the scenarios with 3 VUs. Although the N-EncFL and traditional methods converge at nearly the same time, the EncFL require over 100 iterations to reach the convergence, which is slightly longer than other approaches. However, the gap in learning rate, which is about 15 to 20 iterations, is trivial. It is worth noting that the accuracy of EncFL remains consistent with that of the N-EncFL and CFL, regardless of whether the amount of offloaded data is different. As a result, the proposed framework, which operates on encrypted data, achieves the same accuracy as those of the other benchmarks, i.e., N-EncFL and CFL.

Table 2.3 Simulation results

| **Model** | **2 Vehicle Users** | | | | **3 Vehicle Users** | | | |
|---|---|---|---|---|---|---|---|---|
| | **N-EncFL** | | **EncFL** | | **N-EncFL** | | **EncFL** | |
| | **10%** | **20%** | **10%** | **20%** | **10%** | **20%** | **10%** | **20%** |
| Accuracy | 91.728 | 91.806 | 91.173 | 91.142 | 91.806 | 91.744 | 90.926 | 91.049 |
| Precision | 92.767 | 92.868 | 92.319 | 92.254 | 92.787 | 92.730 | 91.992 | 92.161 |
| Recall | 91.875 | 91.930 | 91.360 | 91.322 | 91.931 | 91.870 | 91.118 | 91.234 |



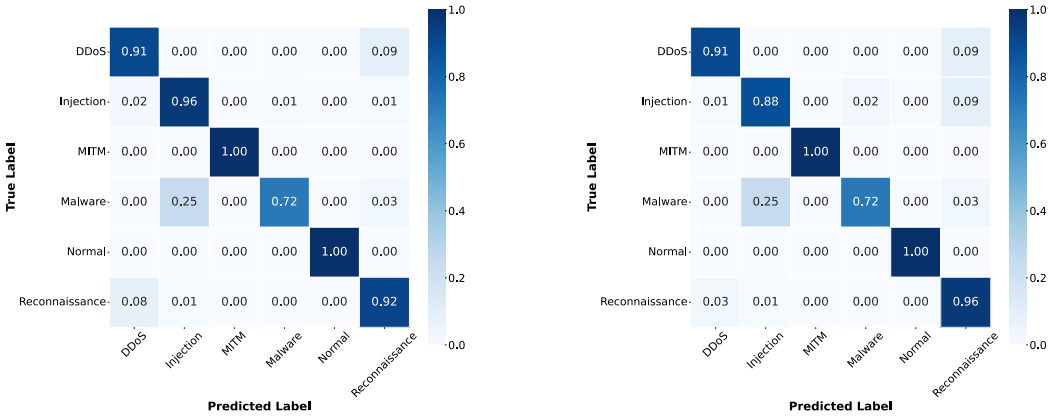

(a) Offloading with 10% non-encrypt data.

(b) Offloading with 10% encrypted data.

Fig. 2.5 Classification results of 2 VUs. The accuracy of all the attacks remains consistent with non-encrypted benchmarks.

### Performance Evaluation

Table 3.4 describes the performance in detecting attacks of two and three VUs in the IoV network. Overall, the accuracy, precision and recall of the two scenarios remain nearly identical. Regarding the different amounts of offloaded data, the results for N-EncFL and EncFL are close to those of other methods. Specifically, even when the data sent is 10% or 20%, the N-EncFL with two or three VUs achieves an accuracy of approximately 91.8%. A similar trend is observed with EncFL, where the accuracy remains consistent regardless of the varying amounts of data offloaded. When comparing the results of EncFL and N-EncFL, we can observe that the accuracy, precision and recall of EncFL are slightly lower than those of the N-EncFL. In detail, the gap between N-EncFL and EncFL with two VUs is from 0.5% to 0.6%. For instance, with 10% offloaded data, EncFL achieves an accuracy of 91.173%, which is 0.55% less than N-EncFL's 91.728%. Additionally, the results of the three VUs show a similar pattern, with EncFL performing 0.6% to 0.8% less than RawFL. However, as observed in Fig. 2.5 and Fig. 2.6, the overall accuracy of 6 classes is nearly the same. The differences primarily lie in the detection of "Injection" and "Reconnaissance", which




(a) Offloading with 20% non-encrypt data.

(b) Offloading with 20% encrypted data.

Fig. 2.6 Classification results of 3 VUs. The same trend is observed with “Reconnaissance” attack, even with the increased offloading data.

accounts for the small gap between N-EncFL and EncFL. Although the accuracy of the “Injection” class of EncFL is less than N-EncFL, EncFL still achieves an 88% detection rate accuracy for the “Injection" attack. As a result, the small gap between N-EncFL and EncFL is acceptable, demonstrating that EncFL can classify each class with a high detection rate.

## 2.5 Summary

In this chapter, we have proposed a novel privacy-preserving FL framework for intrusion detection in IoVs with limited computing resources. The proposed framework enables users to offload data to a centralized server, addressing the computational challenges during local training of the vehicles. To ensure user privacy, homomorphic encryption (HE) is applied to the data before offloading it to the server. The encrypted data is then processed by the training algorithm-based HE, which allows the server to learn from the encrypted data without knowing its content. The proposed framework can protect the privacy of users during the learning process, facilitating the efficient deployment of FL for IDSs in practical IoV networks. The simulation results show that our proposed framework can accurately detect cyberattacks in IoV networks. Although the accuracy of the encrypted neural network is slightly less than that of the raw ones, the gap is acceptable and can be optimized in future works.

# Chapter 3

# Privacy-Preserving Cyberattack Detection in Blockchain-Based IoT Systems Using AI and Homomorphic Encryption

This chapter introduces a novel privacy-preserving cyberattack detection framework for blockchain-based Internet-of-Things (IoT) networks, an emerging approach for future IoT systems. In our approach, artificial intelligence (AI)-driven detection modules are strategically deployed at blockchain nodes to identify real-time attacks, ensuring high accuracy and minimal delay. To achieve this efficiency, the model training is conducted by a cloud service provider (CSP). Accordingly, blockchain nodes send their data to the CSP for training, but to safeguard privacy, the data is encrypted using homomorphic encryption (HE) before transmission. This encryption method allows the CSP to perform computations directly on encrypted data without the need for decryption, preserving data privacy throughout the learning process. To handle the substantial volume of encrypted data, we introduce an innovative packing algorithm in a Single-Instruction-Multiple-Data (SIMD) manner, enabling efficient training on HE-encrypted data. Building on this, we develop a novel deep neural network training algorithm optimized for encrypted data. We further propose a privacy-preserving distributed learning approach based on the FedAvg algorithm, which parallelizes the training across multiple workers, significantly improving computation time. Upon completion, the CSP distributes the trained model to the blockchain nodes, enabling them to perform real-time, privacy-preserved detection. Our simulation results demonstrate that our proposed method can not only mitigate the training time but also achieve detection accuracy that is approximately identical to the approach without encryption, with a gap of around 0.01%.

Additionally, our real implementations on various blockchain consensus algorithms and hardware configurations show that our proposed framework can also be effectively adapted to real-world systems.

## 3.1 System Model

### 3.1.1 Overview of Blockchain-based IoT System

Blockchain is an innovative digital distributed ledger that provides a decentralized, transparent and secure way of storing and managing data across multiple parties. By utilizing the decentralized feature of peer-to-peer networks, each participant can maintain a copy of the ledger, therefore eliminating the need for centralized authority. However, when integrating blockchain to IoT systems, many devices are constrained by limited resources, which makes it challenging for them to hold a copy of the ledger or engage in the mining process, such as validating or publishing new blocks [9]. These resource limitations necessitate tailored approaches for integrating IoT devices (e.g., gateways and sensors) into blockchain networks, ensuring that they can participate effectively without being overburdened. In such scenarios, IoT devices may focus on transmitting data to more powerful nodes in the network (e.g. blockchain mining nodes), which handle the extensive computational tasks associated with blockchain operations [9]. This approach allows IoT systems to function based on blockchain, harnessing its decentralization, security, and transparency in IoT data management while maintaining efficiency and scalability despite the constraints of individual devices.

There are several types of blockchain networks, each tailored to meet the specific needs of different IoT applications, including Public Blockchain, Private Blockchain, and Consortium Blockchain [9]. The public blockchain permits any user to participate in the network, in which the users are required to pay an incentive fee to send transactions. In contrast, private and consortium blockchains restrict access to only authorized nodes while removing the need for transaction fees, enabling the deployment of applications for confidential purposes. Although the type of employed blockchain depends on the purposes of IoT applications, blockchain-based IoT systems still operate using the same approach. In general, the IoT devices send the transactions consisting of IoT data to the blockchain nodes for the mining process. Based on different consensus mechanisms (e.g. PoW, PoS, PBFT), the validators (miners) will store transactions into a block and compete to become the block leaders [74]. Once the new block is verified on the chain, it is linked to the previous block via the hash value of the block's header, making it nearly impossible to alter the recorded information in the block and thereby ensuring the security and immutability of the blockchain [10]. In

summary, the ability of IoT devices to participate in the blockchain enables IoT systems to achieve a more secure, immutable, and decentralized network.

### 3.1.2 Privacy-Preserving Cyberattack Detection in Blockchain Networks

The proposed privacy-preserving cyberattack detection system is illustrated in Fig. 3.1. The system consists of a CSP consisting of a master node (MN) along with $M$ worker nodes (WNs), and $N$ blockchain nodes (BNs). In our framework, we consider an IoT network operated by an IoT Service Provider (IoTSP). Similar to [75], the IoTSP deploys a private blockchain network to manage the IoT service and transactions, thereby maintaining trust between blockchain nodes and IoT endpoints. In this regard, cyberattacks on blockchain nodes can potentially take control of the transactions in blockchain networks, and thus Cyberattack Detection Modules (CDMs) can be deployed at the blockchain nodes to detect and prevent cyberattacks. Nevertheless, deploying the CDMs on the blockchain nodes for real-time detection is challenging. As mentioned in Section 3.1.1, since the blockchain nodes must perform various functions to maintain the blockchain (e.g., mining, validating, verifying, and storing blocks), training the CDMs on these nodes is inefficient due to extensive computational demand of training task. Although the CSP with extensive cloud computing resources is an effective solution to support the IoTSP within the training and deployment of the CDM, processing the CDMs externally on a cloud server raises significant concerns regarding user data privacy. To address this, our proposed framework employs the HE to protect user privacy while ensuring accurate deployment of CDMs on blockchain nodes. Specifically, our privacy-preserving cyberattack detection framework includes three phases: data offloading phase, encrypted data training phase, and real-time detection phase, as illustrated in Fig. 3.1.

In particular, as shown in Fig. 3.1, in the offloading phase, the local dataset of each BN will be offloaded to the CSP for further processing. However, before transmitting data to the CSP, BNs will generate HE key pairs and use them to encrypt the uploading data. Once the CSP receives the encrypted data from blockchain nodes, it will assemble it into a large encrypted dataset and then initialize a learning model specifically designed for cyberattack detection to prepare for the next phase. The CSP starts the training phase by initially evaluating the number of available WNs within the cloud environment. Based on this assessment, the MN strategically partitions the full encrypted dataset into multiple segments and then distributes each across the predefined WNs. In this way, the computing resources of the active WNs are leveraged to maximize the training efficiency of the large encrypted dataset, reducing

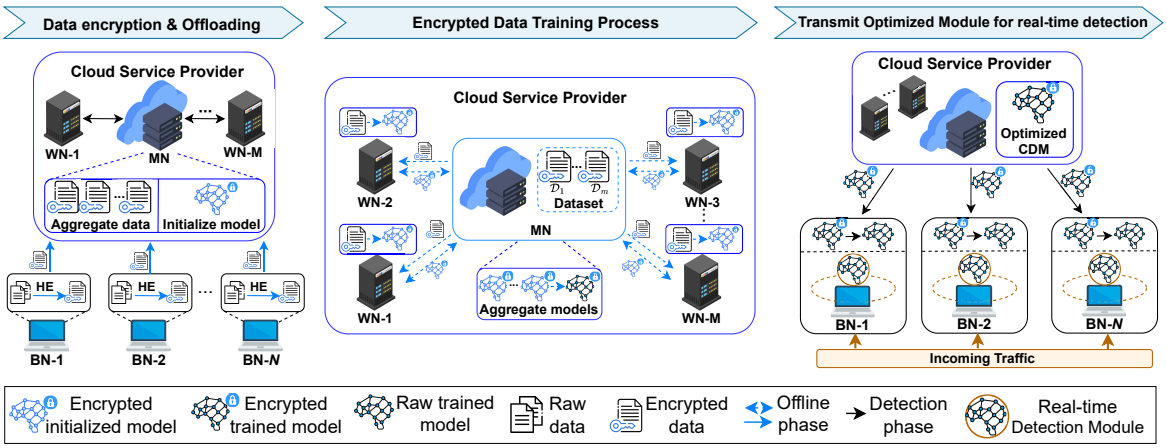



Fig. 3.1 The proposed privacy-preserving cyberattack detection framework includes three phases: data encryption and offloading, encrypted data training, and real-time detection. The CSP operates as a network security service to ensure the security of $N$ blockchain nodes (BNs).

the computational overhead challenge of HE-encrypted data training. Upon receiving the corresponding encrypted partition, each WN will use the initialized learning model to train its assigned segment using our proposed training algorithm for encrypted data, detailed in Section 3.2.1. After completing each learning round, the WNs will send the encrypted trained models to the MN, which aggregates them into an updated encrypted model (the aggregation of the encrypted models is detailed in Section 3.2.2). This updated model is then transmitted back to WNs to begin the next learning round. The CSP maintain this iterative process until the considered learning model converges or the predetermined number of learning rounds is reached, ensuring the model is fully optimized for the next phase. During the real-time cyberattack detection phase, the CSP sends the optimized encrypted model back to the BNs for local deployment. Once the BNs receive the model, they will decrypt the model to form a real-time detection module which can detect and prevent incoming attack traffic to the blockchain network.

## 3.2 The Privacy-Preserving Distributed Learning

As we described in Section 3.1, the CSP is responsible for training the learning model-based HE with a large encrypted dataset collected from blockchain nodes. However, as discussed in Section 1.3.2, training deep learning models on HE-encrypted data presents two major challenges: the lack of optimized training algorithms specifically designed for encrypted data and the significant computational overhead that results in extensive training time. To

**Algorithm 2** Pack1D for packing 1D vector

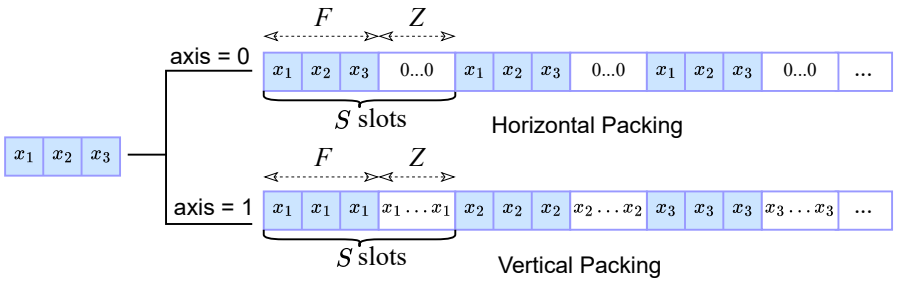
**Input:** $x, S, \mathscr{B}, \bar{x}_{\text{axis}}$, with $x = [x_1, x_2, \ldots, x_n]$ and $\bar{x}_{\text{axis}} \in \{0, 1\}$
**Output:** $\bar{x}$
Initialize $step = \frac{\mathscr{B}}{S}$
**if** $\bar{x}_{\text{axis}} = 0$ **then**
    $x = \mathsf{ZeroPad}(x, S)$
    $\bar{x} = \mathsf{Replicate}(x, step)$ which $\bar{x}_{i+j \cdot n} = x_i,\ i \in \{1, 2, \ldots, n\}$ and $j \in \{0, 1, \ldots, step - 1\}$
**else**
    $\bar{x} = \mathsf{Repeat}(x, S)$ which $\bar{x}_{(i-1) \cdot S + j} = x_i,\ i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, S\}$
**end if**
$\bar{x} = \mathsf{ZeroPad}(\bar{x}, \mathscr{B})$ S
**return** $\bar{x}$

address these problems, we initially propose an effective training algorithm for the deep neural network with HE-encrypted data. Following that, we design the distributed training approach to optimize the processing time of our training algorithm.

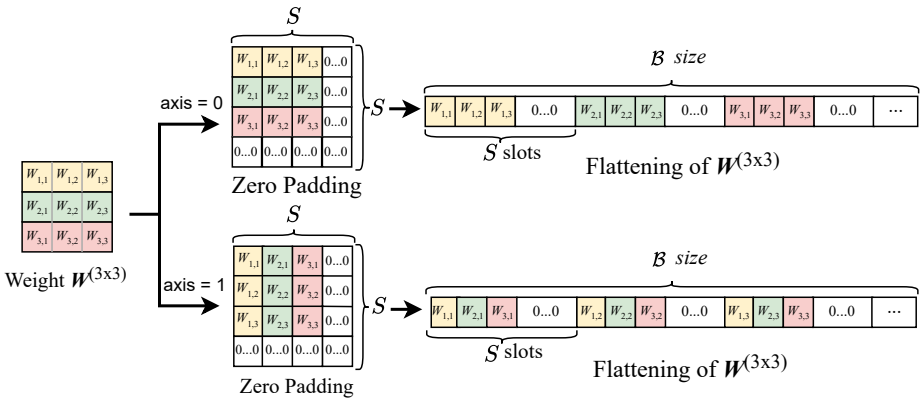
### 3.2.1 The Proposed Training Process of Deep Neural Network for HE-Encrypted Data

#### The Proposed Packing Methods

For efficient homomorphic evaluation during the training task, we first develop two alternative packing approaches called Pack1D and Pack2D. To be more specific, Pack1D is applied to process one-dimensional (1D) data and then produce the output in corresponding $\bar{x}_{\text{axis}}$, in which $\bar{x}_{\text{axis}}$ is the predefined axis of the output (e.g., $\bar{x}_{\text{axis}} = 0$ for horizontal axis and $\bar{x}_{\text{axis}} = 1$ for vertical axis). It is worth noting that in HE, a plaintext contains multiple segments, with the number of slots $S$ in each segment, and each slot holds an individual value. As illustrated in Algorithm 2, Pack1D takes the input consisting of a 1D array, slot size of a segment, size of ciphertext, and the considered axis of output, which are respectively defined as $x$, $S$, $\mathscr{B}$, and $\bar{x}_{\text{axis}}$. As described in Fig. 3.2(a), the axis for encryption (i.e., horizontal or vertical) is first determined. Regarding the horizontal axis, the sample is zero-padded to match with a plaintext slot size $S$, where $S = \lfloor \sqrt{\mathscr{B}} \rfloor$ and $S = F + Z$, with $F$ being the number of values in the sample and $Z$ being the number of added zeros. Notably, $F$ is typically less than $S$ as network traffic data, the focus of this paper, often has a limited number of features. Then, it is replicated $\mathscr{B}/S$ times to fit with the size of the ciphertext. Otherwise, if the axis is




(a) 1D packing for a vector




(b) 2D packing for a matrix

Fig. 3.2 Illustration of the proposed alternative packing method, including (a) 1D packing and (b) 2D packing. The proposed methods are the pre-processing of HE, which fits the elements of data (i.e., vector and matrix) into the slots of a ciphertext in both normal (axis=0) and transpose (axis=1) manners. Therefore, this enables efficient encrypted matrix/vector multiplications during the training task.

vertical, each element of the sample is repeated $S$ times, forming a new array, which is then zero-padded to fit with the size of the ciphertext.

The proposed Pack2D algorithm is described in Algorithm 3, which is applied to pack a two-dimensional (2D) matrix into a 1D vector. Similar to Pack1D, Pack2D initially determines the axis for encryption. If the axis is vertical, the matrix is transposed; otherwise, it remains unchanged. Subsequently, the matrix is zero-padded to a square matrix with the size of $S \times S$. This square matrix is then flattened by concatenating its rows to form a 1D vector. For more clarity, the implementation of our 2D packing method is shown in Fig. 3.2(b).

To perform encryption based on our proposed packing methods, we consider a dataset $\mathscr{D} = (\mathbf{x}_i, \mathbf{y}_i)$ consisting of $I$ samples where $i \in \{1, \ldots, I\}$. Here, $\mathbf{x}_i$ and $\mathbf{y}_i$ are, respectively, the training samples and labels, in which $\mathbf{x}_i$ is a feature vector of i-th sample attached with a one-

**Algorithm 3** Pack2D for packing 2D matrix

**Input:** $X, S, \mathscr{B}, \bar{x}_{\text{axis}}$, with $X \in \mathbb{R}^{m \times n}$ and $\bar{x}_{\text{axis}} \in \{0,1\}$
**Output:** $\bar{x}$
**if** $\bar{x}_{\text{axis}} = 1$ **then**
    $\bar{X} = \mathsf{Transpose}(X)$
**else**
    $\bar{X} = X$
**end if**
$\bar{X} = \mathsf{ZeroPad}(\bar{X}, S)$ which $m \times n = S^2$
$\bar{x} = \mathsf{Flatten}(\bar{X})$
**return** $\bar{x}$

hot encoded label $\mathbf{y}_i$. Based on the aforementioned HE operations, the key pair, including the public key and secret key, are used to generate the encrypted dataset $\hat{\mathscr{D}}$. Before the encryption of the dataset, we apply the 1D packing algorithm, described in Algorithm 2 as the preprocessing step for each sample. Therefore, the encryption process of encrypted feature vectors and encrypted labels can be denoted as:

$$\hat{\boldsymbol{x}}_i = \mathsf{Enc}\big(\mathsf{Pack1D}(\mathbf{x}_i, \bar{x}_{\text{axis}}, S, \mathscr{B}), \mathsf{pk}\big), \tag{3.1}$$

$$\hat{\boldsymbol{y}}_i = \mathsf{Enc}\big(\mathsf{Pack1D}(\mathbf{y}_i, \bar{y}_{\text{axis}}, S, \mathscr{B}), \mathsf{pk}\big), \tag{3.2}$$

where $\hat{\boldsymbol{x}}_i$ and $\hat{\boldsymbol{y}}_i$ ($\forall i \in \{1, \ldots, I\}$) are the encrypted feature vector and its attached encrypted label of the dataset $\hat{\mathscr{D}}$, respectively. This encrypted dataset will be the input for a deep neural network consisting of $K$ layers, in which its plain form can be represented as:

$$h^{(k)} = \sigma(W^{(k)} \cdot h^{(k-1)} + b^{(k)}), \quad k \in \{1, \ldots, K\}, \tag{3.3}$$

where $W^{(k)}$, $h^{(k)}$, $b^{(k)}$, $\sigma$ are parameters of the neural network consisting of weights, the output of layer $k$, biases, and activation function, respectively. Here, $h^{(0)}$ is the input feature vector $x_{in}$ while $h^{(K)}$ is the output of the neural network $x_{out}$. In conventional deep learning, these parameters will be applied to produce the optimized output $x_{out}$ via the training process. Here, to process HE-encrypted data, the weights and biases are also encrypted to homomorphic form. Before performing encryption, the weights will be preprocessed by our proposed 2D packing algorithm, which is illustrated in Algorithm 3. In particular, the encryption process is denoted as:

$$\hat{\boldsymbol{W}}^{(k)} = \mathsf{Enc}\big(\mathsf{Pack2D}(W^{(k)}, \bar{W}^{(k)}_{\text{axis}}, S, \mathscr{B}), \mathsf{pk}\big), \tag{3.4}$$

where $\hat{\boldsymbol{W}}^{(k)}$ is the encrypted weight matrix of layer $k$. To allow the multiplication of the weight matrix in encrypted form, the axis of $\hat{\boldsymbol{W}}^{(k)}$ is varied based on k-th layer. In particular, the value of $\bar{W}_{\text{axis}}$ can be defined as:

$$\bar{W}_{\text{axis}}^{(k)} = \begin{cases} 0, & \text{if } k \; mod \; 2 = 1, \\ 1, & \text{if } k \; mod \; 2 = 0, \end{cases} \tag{3.5}$$

An axis value of 0 enables the computation along the row-based axis, whereas the axis value of 1 facilitates the column-based processing, as depicted in Fig. 3.2. Following that, the biases will be processed by applying the 1D packing algorithm before the encryption, which is denoted as:

$$\hat{\boldsymbol{b}}^{(k)} = \mathsf{Enc}\big(\mathsf{Pack1D}(b^{(k)}, \bar{b}_{\text{axis}}^{(k)}, S, \mathscr{B}), \mathsf{pk}\big), \tag{3.6}$$

where $\hat{\boldsymbol{b}}^{(k)}$ is the encrypted bias vector of layer $k$. Thus, the output of the $\hat{\boldsymbol{W}}^{(k)}$ is the ciphertext on the orthogonal axis of its encrypted weight. For instance, the encrypted weight matrix packed on a row-based axis will generate output on a column-based axis. Hence, the axis of encrypted bias is calculated as:

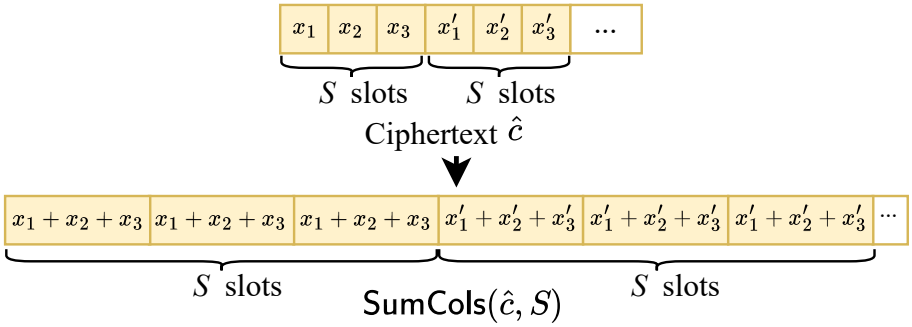
$$\bar{b}_{\text{axis}}^{(k)} = 1 - \bar{W}_{\text{axis}}^{(k)}. \tag{3.7}$$

By leveraging our proposed packing methods to process the encrypted input and encrypted parameters of the neural network in an alternative row-column approach, it is feasible to perform the training task on HE-encrypted data in a SIMD manner, which will be described in the following subsection.

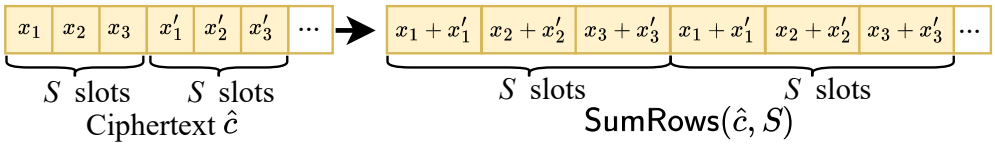
### The Proposed Training Process

During the training task, while performing the homomorphic evaluation of the encrypted parameters, we utilize the SumCols and SumRows algorithms in [1]. These algorithms take input as an encrypted vector and $S$ slot size to allow the sum operation of multiple segments on a different axis (i.e., rows or columns) of the encrypted vector, which is further described in Fig 3.3. After finishing generating the encrypted parameters of the neural network, the training process starts by initially forwarding the encrypted data through the encrypted parameters. In the feed-forward process, the output at layer $k$ of the neural network, denoted as $\hat{\boldsymbol{h}}^{(k)}$, can be calculated as:

$$\hat{\boldsymbol{h}}^{(k)} = \hat{\sigma}^{(k)} \left( \hat{\boldsymbol{W}}^{(k)} \hat{\otimes} \hat{\boldsymbol{h}}^{(k-1)} \hat{\oplus} \hat{\boldsymbol{b}}^{(k)} \right), \tag{3.8}$$




(a) Illustration of SumCols(.) algorithm. It first performs summation on all elements within each segment of the ciphertext. Then, the summation result is fit into each slot of the considered segment.




(b) Illustration of SumRows(.) algorithm. It first performs element-wise summation from different segments. This results in new segments with the same data where each slot of the segments is the combined sum of element $x_i$ from the original segments.

Fig. 3.3 Illustration of SumCols(.) and SumRows(.) algorithms on a ciphertext with multiple segments, each having $S$ slots size where each slot represents a distinct element $x_i$ [1].

where $\hat{\boldsymbol{W}}^{(k)}$ and $\hat{\boldsymbol{b}}^{(k)}$ represent the encrypted weight matrix and encrypted bias vector of layer $k$, as explained in (3.4) and (3.6), respectively. $\hat{\otimes}$ and $\hat{\oplus}$ are, respectively, the homomorphic multiplication and addition for encrypted parameters of the neural network, which can be defined as:

$$\hat{\boldsymbol{W}} \hat{\otimes} \hat{\boldsymbol{x}} = \begin{cases} \mathsf{SumCols}\big(\mathsf{Mult}(\hat{\boldsymbol{W}}, \hat{\boldsymbol{x}}), S\big), & \text{if } \hat{\boldsymbol{W}}_{\text{axis}} = 0 \\ \mathsf{SumRows}\big(\mathsf{Mult}(\hat{\boldsymbol{W}}, \hat{\boldsymbol{x}}), S\big), & \text{if } \hat{\boldsymbol{W}}_{\text{axis}} = 1 \end{cases} \tag{3.9}$$

$$\hat{\boldsymbol{W}} \hat{\oplus} \hat{\boldsymbol{x}} = \mathsf{Add}(\hat{\boldsymbol{W}}, \hat{\boldsymbol{b}}), \tag{3.10}$$

where $\hat{\oplus}$ and $\hat{\otimes}$ are basically based on the Add(.), Mult(.), SumCols(.), and SumRows algorithms. To be more specific, after multiplying the input ciphertext $\hat{\boldsymbol{x}}$ with $\hat{\boldsymbol{W}}$, the $\hat{\otimes}$ proceeds summation based on the axis of $\hat{\boldsymbol{W}}$, which is illustrated in (3.9). The sum ciphertext is then applied Add(.) with $\hat{\boldsymbol{b}}$ to form the final output ciphertext. Subsequently, this ciphertext is passed through $\hat{\sigma}$, which illustrates the polynomial approximation of the activation function. As HE only supports homomorphic evaluations (e.g., addition, multiplication), the conventional activation function, such as *ReLU* or *Tanh*, may not be

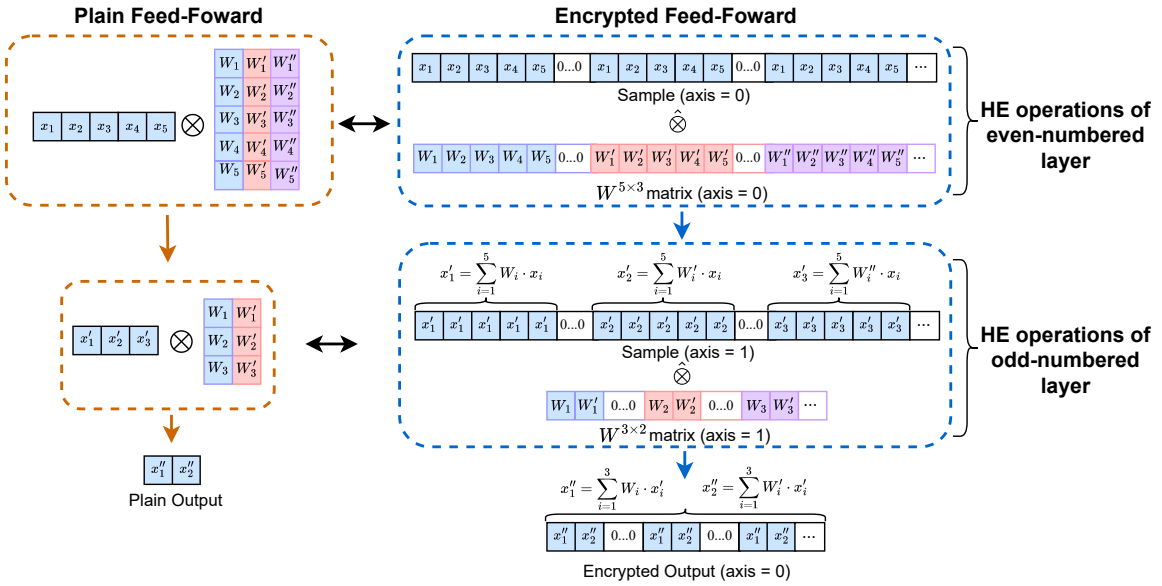



Fig. 3.4 Implementation of encrypted feed-forward in a neural network with two layers. The weight matrix of the neural network layer is packed and encrypted based on the $k$-th order of the considered layer, regarding normal processing (axis=0) or transpose processing (axis=1). Therefore, through the HE multiplication described in (3.9), the input ciphertext, initially packed along axis=0, can alternatively fit into the encrypted neural network layers.

efficient for HE operations [68]. Accordingly, in this paper, we employ the Chebyshev polynomial method [68] to approximate the *Swish* (SiLU) function [69]. The SiLU is chosen for its advantage in mitigating the "dying *ReLU*" problem. To be more specific, the general forward process of our proposed training algorithm is described in Fig. 3.4.

In the backpropagation process, we leverage the Stochastic Gradient Descent (SGD) for mini-batch and the Mean Squared Error (MSE) loss to enable the back-propagation of the encrypted data. Given a batch with $B$ pairs of predicted label $\hat{\boldsymbol{a}}_i$ and true label $\hat{\boldsymbol{y}}_i$, the MSE loss for optimizing encrypted model parameters during training can be defined as:

$$\hat{\mathscr{L}}_{MSE} = \mathsf{Mult}\Big(\sum_{i=1}^{B} \mathsf{Square}\big(\mathsf{Sub}(\hat{\boldsymbol{y}}_i, \hat{\boldsymbol{a}}_i)\big), \frac{1}{B}\Big), \tag{3.11}$$

in which $\mathsf{Square}(\hat{\boldsymbol{c}}) = \mathsf{Mult}(\hat{\boldsymbol{c}}, \hat{\boldsymbol{c}})$. Following that, the gradient of the encrypted MSE loss can be denoted as:

$$\nabla\hat{\mathscr{L}}_{MSE,i} = \frac{\partial\hat{\mathscr{L}}_{MSE}}{\partial\hat{\boldsymbol{a}}_i} = \mathsf{Mult}\Big(\mathsf{Sub}(\hat{\boldsymbol{y}}_i, \hat{\boldsymbol{a}}_i), \frac{2}{B}\Big). \tag{3.12}$$

**Algorithm 4** Iterative Training Algorithm for HE-Encrypted Data

**Input:** Initialize the encrypted data $\hat{\mathscr{D}} = (\hat{\boldsymbol{x}}_i, \hat{\boldsymbol{y}}_i)$ based on (3.1), (3.2); $i \in I$

Initialize the encrypted parameters $\hat{\theta} = (\hat{\boldsymbol{W}}^{(k)}, \hat{\boldsymbol{b}}^{(k)})$ based on (3.4), (3.6); $k \in K$

Initialize $T$ training rounds, learning rate $\eta$, and the HE parameters including (pk,sk), $S$, $\mathscr{R}$

**for** $t = 0 \rightarrow T-1$ **do**

Compute the output $\hat{\boldsymbol{a}}^{(k)}$ over layer $k$ using (3.9), (3.10):

$$\hat{\boldsymbol{a}}_t^{(k)} = \hat{\sigma}^{(k)}\left(\hat{\boldsymbol{W}}_t^{(k)} \hat{\otimes} \hat{x}^{(k-1)} \hat{\oplus} \hat{\boldsymbol{b}}_t^{(k)}\right)$$

Compute gradient $\hat{\mathscr{L}}_{MSE,i}^t$ for all mini-batch of $B$ samples in $\hat{\mathscr{D}}$:

$$\nabla\hat{\mathscr{L}}_{MSE,i}^t = \mathsf{Mult}\big(\mathsf{Sub}(\hat{\boldsymbol{y}}_i, \hat{\boldsymbol{a}}_{i,t}), \tfrac{2}{B}\big)$$

Compute gradient of layer $K$:

$$\nabla\hat{\boldsymbol{g}}_{i,t}^{(K)} = \mathsf{Mult}\Big(\nabla\hat{\mathscr{L}}_{MSE,i}^t, \tfrac{\partial\hat{\sigma}^{(K)}}{\partial\hat{\boldsymbol{x}}_i^{(K-1)}}\Big)$$

Backward the gradient for $k$ layers:

$$\nabla\hat{\boldsymbol{g}}_{i,t}^{(k)} = \mathsf{Mult}(\nabla\hat{\boldsymbol{g}}_{i,t}^{(k+1)}, \hat{\boldsymbol{W}}_t^{(k+1)})$$

Produce final gradient $\nabla\hat{\boldsymbol{g}}_{i,t}^{(k)}$ using (3.15)

Compute:

$$\nabla\hat{\boldsymbol{W}}_t^{(k)} = \textstyle\sum_{i=1}^{B} \mathsf{Mult}(\hat{\boldsymbol{x}}_i^{(k-1)}, \nabla\hat{\boldsymbol{g}}_{i,t}^{(k)})$$

and:

$$\nabla\hat{\boldsymbol{b}}_t^{k} = \textstyle\sum_{i=1}^{B} \nabla\hat{\boldsymbol{g}}_{i,t}^{(k)}$$

Update the $\hat{\theta}$ by SGD for encrypted data:

$$\hat{\boldsymbol{W}}_t^{(k)} \leftarrow \mathsf{Sub}\Big(\hat{\boldsymbol{W}}_t^{(k)}, \mathsf{Mult}\big(\eta, \nabla\hat{\boldsymbol{W}}_t^{(k)}\big)\Big)$$

and:

$$\hat{\boldsymbol{b}}_t^{(k)} \leftarrow \mathsf{Sub}\Big(\hat{\boldsymbol{b}}_t^{(k)}, \mathsf{Mult}\big(\eta, \nabla\hat{\boldsymbol{b}}_t^{(k)}\big)\Big)$$

Bootstrap the encrypted weights and biases:

$$\hat{\boldsymbol{W}}_t^{(k)} = \mathsf{Bootstrap}(\hat{\boldsymbol{W}}_t^{(k)})$$

and:

$$\hat{\boldsymbol{b}}_t^{(k)} = \mathsf{Bootstrap}(\hat{\boldsymbol{b}}_t^{(k)})$$

**end for**

The calculated gradient $\nabla\hat{\mathscr{L}}_{MSE,i}$ is then utilized to calculate the gradients for $k$ encrypted layers. Given that $K$ is the latest layer of the neural network, the backward process starts by initially calculating the encrypted gradient of layer $K$:

$$\nabla\hat{\boldsymbol{g}}_i^{(K)} = \mathsf{Mult}\Big(\nabla\hat{\mathscr{L}}_{MSE,i}, \frac{\partial\hat{\sigma}^{(K)}}{\partial\hat{\boldsymbol{h}}_i^{(K-1)}}\Big). \tag{3.13}$$

Regarding the plain back-propagation, the gradient backward process requires the transpose matrix multiplication, which is challenging for the current HE-based neural networks. However, by employing our packing methods described in Algorithm 2 and Algorithm 3, we can alternatively perform transpose multiplication between ciphertexts. Accordingly, the $\nabla\hat{\boldsymbol{g}}_i^{(K)}$ obtained from (3.13) is then backward to the previous $K-1$ layers. In particular, the backward of $\nabla\hat{\boldsymbol{g}}_i^{(K)}$ can be defined as:

$$\nabla\hat{\boldsymbol{g}}_i^{(k)} = \mathsf{Mult}(\nabla\hat{\boldsymbol{g}}_i^{(k+1)}, \hat{\boldsymbol{W}}^{(k+1)}), \tag{3.14}$$

in which $\nabla\hat{\boldsymbol{g}}_i^{(k)}$ is then applied to the summation algorithms to produce the final gradient for layer $k$:

$$\nabla\hat{\boldsymbol{g}}_i^{(k)} = \begin{cases} \mathsf{SumRows}\big(\nabla\hat{\boldsymbol{g}}_i^{(k)}, S\big), & \text{if } \bar{W}_{\text{axis}}^{(k+1)} = 0, \\ \mathsf{SumCols}\big(\nabla\hat{\boldsymbol{g}}_i^{(k)}, S\big), & \text{if } \bar{W}_{\text{axis}}^{(k+1)} = 1. \end{cases} \tag{3.15}$$

Subsequently, the $\nabla\hat{\boldsymbol{g}}^{(k)}$ is used to generate the $\nabla\hat{\boldsymbol{W}}^{(k)}$ and $\nabla\hat{\boldsymbol{b}}^{(k)}$ of layer $k$ which is calculated as:

$$\nabla\hat{\boldsymbol{W}}^{(k)} = \sum_{i=1}^{B} \mathsf{Mult}(\hat{\boldsymbol{h}}_i^{(k-1)}, \nabla\hat{\boldsymbol{g}}_i^{(k)}), \tag{3.16}$$

$$\nabla\hat{\boldsymbol{b}}^{(k)} = \sum_{i=1}^{B} \nabla\hat{\boldsymbol{g}}_i^{(k)}, \tag{3.17}$$

After generating the gradient for weight and bias, the SGD momentum is employed to update the parameters during the training process. The SGD update for encrypted parameters can be defined as:

$$\hat{\boldsymbol{W}}^{(k)} \leftarrow \mathsf{BootStrap}\Big(\mathsf{Sub}\Big(\hat{\boldsymbol{W}}^{(k)}, \mathsf{Mult}\big(\eta, \nabla\hat{\boldsymbol{W}}^{(k)}\big)\Big)\Big), \tag{3.18}$$

$$\hat{\boldsymbol{b}}^{(k)} \leftarrow \mathsf{BootStrap}\Big(\mathsf{Sub}\Big(\hat{\boldsymbol{b}}^{(k)}, \mathsf{Mult}\big(\eta, \nabla\hat{\boldsymbol{b}}^{(k)}\big)\Big)\Big). \tag{3.19}$$

**Algorithm 5** Privacy-Preserving Distributed Learning-Enabled Cyberattack Detection Framework

**for** $\forall n \in N$ **do**
  $N$ Blockchain nodes generate the keypairs: $\mathsf{sk}_n = \mathsf{SKGen}(n)$ and $\mathsf{pk}_n = \mathsf{PKGen}(\mathsf{sk}_n)$
  Extract $\mathscr{D}_n$ and process it by using Algorithm 2
  Generate the encrypted data $\hat{\mathscr{D}}_n = \mathsf{Enc}(\mathsf{pk}_n, \mathscr{D}_n)$
  Send the encrypted data $\hat{\mathscr{D}}_n$ to the CSP
**end for**
CSP combines received data into an encrypted dataset $\hat{\mathscr{D}}$
CSP initializes the deep learning model $\Theta$ and training parameters $T$, $B$
Pack $\Theta$ by using Algorithm 3 and generate the encrypted model $\hat{\Theta}$ where $\hat{\Theta} = \mathsf{Enc}(\mathsf{pk}_n, \Theta)$
Distribute $\hat{\Theta}$, $B$, and $T$ to $M$ WNs
Split the encrypted dataset $\hat{\mathscr{D}}$ to $M$ partitions $\hat{\mathscr{D}}_m$ and assign each to respective WN-$m$
**while** $t \leq T$ or training process does not converge **do**
  **for** $\forall m \in M$ **do**
    WN-$m$ calculates the trained parameters $\hat{\Theta}_m^t$ using Algorithm 4
    WN-$m$ send their $\hat{\Theta}_m^t$ to the MN
  **end for**
  MN produces the encrypted aggregated model $\hat{\Theta}_a^{(t+1)}$
  Send the updated model $\hat{\Theta}_a^{(t+1)}$ back to $M$ WNs
**end while**
CSP transmits the optimized model $\hat{\Theta} = \hat{\Theta}^{(T)}$ to $N$ BNs
**for** $n \leq N$ **do**
  Decrypt the model $\Theta = \mathsf{Dec}(\hat{\Theta}, \mathsf{sk}_n)$
  Predict incoming data based on the optimized model $\Theta$ in real-time
**end for**

in which the SGD optimizer for encrypted data uses the basic HE operations mentioned in 1.2.3 with learning rate $\eta$ to update the weights and biases of the HE-based neural network iteratively. In addition, the bootstrapping mechanism is applied to the encrypted parameters to reduce the noise of ciphertext, allowing additional operations on encrypted data across multiple training rounds. To summarize, the proposed training algorithm for HE-encrypted data is described in Algorithm 4. This iterative training method will be employed in our proposed privacy-preserving distributed learning for cyberattack detection, which is described as follows.

### 3.2.2 Implementation of Distributed Cloud-Native Learning

As mentioned in Section 3.1.2, the BNs enter the offloading phase by initially generating the HE key pairs, which consist of the secret key $\mathsf{sk}_n$ and the public key $\mathsf{pk}_n$. The secret key is securely stored within the respective BNs, while the public key is broadcasted to the CSP and other nodes in the blockchain network. Besides validating transactions, BN-$n$ encrypt its local dataset $\mathscr{D}_n$ to $\hat{\mathscr{D}}_n$ by employing mechanisms in (3.1) (3.2). Subsequently, the encrypted data is transmitted to the CSP, where it is combined to create a large encrypted dataset $\hat{\mathscr{D}}$. After that, the CSP generates the deep-learning model with parameters $\theta$ and encrypts it to $\hat{\theta}$ by using equations (3.4) and (3.6).

To start the encrypted data training phase, the CSP apply our proposed privacy-preserving distributed learning (PPDiL) by first determining the training parameters, including $T$ training rounds and batch size $B$. Concurrently, it leverages the MN and $M$ WNs within its cluster environment, which employs the computation from multiple workers while maintaining the MN as an aggregate point. Accordingly, the MN distributes the training parameters and encrypted model to each WN-$m$, with the local batch size $\lfloor \frac{B}{M} \rfloor$. The MN then divides the encrypted dataset $\hat{\mathscr{D}}$ into $M$ encrypted partitions $\hat{\mathscr{D}}_m$ and assigns each partition to respective WN-$m$. At training each round $t$, WN-$m$ employs the proposed training algorithm described in Algorithm 4 to perform encrypted training on the corresponding $\hat{\mathscr{D}}_m$ and produce the encrypted trained model $\hat{\Theta}_m^t$. Afterwards, the workers transmit the $\hat{\Theta}_m^t$ to the MN, which then starts the aggregation by utilizing the FedAvg algorithm [32], which is modified for encrypted data. In particular, it can be defined by:

$$\hat{\Theta}_a^{(t+1)} = \mathsf{Mult}\Big(\frac{1}{M}, \sum_{m=1}^{M} \hat{\Theta}_m^t\Big). \tag{3.20}$$

At this point, the aggregated parameters $\hat{\Theta}_a^{(t+1)}$ are sent back to the WNs, which are subsequently updated to the workers' encrypted model for the next training round. This training process is iteratively maintained until the aggregated model converges and the CSP obtains the optimal parameters.

After retrieving the optimized encrypted model $\hat{\Theta} = \hat{\Theta}^{(T)}$, the real-time cyberattack detection phase will commence. In this stage, the CSP transmit the encrypted model $\hat{\Theta}$ to the BNs for local deployment. After receiving $\hat{\Theta}$, BN-$n$ decrypt $\hat{\Theta}$ to $\Theta$ using its $\mathsf{sk}_n$ and operate $\Theta$ as a cyberattack detection module to classify the incoming network traffic in real-time. Generally, the implementation of our proposed distributed learning algorithm is detailed in Algorithm 5.

## 3.3 Performance Evaluation

### 3.3.1 Simulation Setup and Evaluation Metrics

#### Dataset and PPDiL Parameters

To evaluate our proposed framework, we use the Blockchain network traffic data from the BNAT dataset [61], which is designed for cyberattack detection in blockchain-based IoT networks. The dataset is collected via the IoT gateways and the blockchain nodes, with IoT gateways transmitting IoT transactions to the blockchain nodes, thereby making it ideal for our considered system. The BNAT dataset contains normal traffic and four types of Blockchain network attacks, including Denial of Services (DoS), Brute Password (BP), Flooding of Transaction (FoT), and Man-in-the-Middle (MitM). During the preprocessing, we downsampled the dataset into 10,000 samples, with 2,000 samples for each class. The dataset is then nominalized and min-max scaled within the range of (0,1). After that, we divide the dataset into training and testing sets with a ratio of 80:20, in which the training set is HE-encrypted, and the testing set is configured for two scenarios: non-encrypted inference and encrypted inference. Regarding the learning model configuration, we design a deep neural network containing an input layer, 2 hidden layers and an output layer. The number of neurons in the layers is 21, 32, 16, and 5, respectively. Apart from the input layer, each layer is connected to the polynomial approximation of the SiLU activation function. Following that, we utilize the OpenFHE library [76] for HE and the Pytorch library for designing neural networks.

Due to the privacy-preserving distributed learning (PPDiL) setup, we select the Flower open-source [77] for the experiment. We configure the CSP to have one master node and a maximum of five worker nodes operating by six different workstations within a local cluster environment. Three of these workstations are equipped with Intel Xeon E-2288G @3.7GHz processors with 8 cores. The remaining three workstations are powered by Intel Xeon Gold 6238R @2.2GHz processors with 28 cores (26 cores enabled). As described in Fig. 3.5, the MN and WNs are deployed by the Google Remote Procedure Call (gRPC) server-client model, and their connections are maintained by TCP protocol with a connection timeout set at 4600 seconds. This cluster environment is employed to apply our proposed PPDiL for HE-encrypted data with the parameters illustrated in Table 3.1.

During the simulation, we evaluate our proposed distributed learning framework with different numbers of workers (i.e., two workers to five workers) and a centralized approach regarding processing time. Similar to [24], we consider the training with non-encrypted

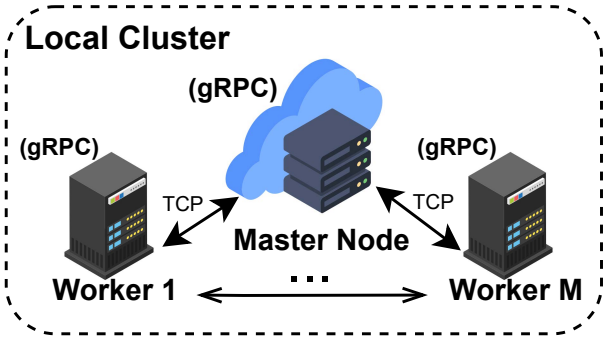



Fig. 3.5 The experiment setup of CSP environment.

Table 3.1 Parameters setting

| HE-Training Parameters | Values |
|---|---|
| Training rounds | $T = 30$ |
| Initial batch size | $B = 128$ |
| Learning rate | $\eta = 0.9$ |
| HE-Ring dimension | $\mathscr{R} = 2^{11}$ |
| HE-ciphertext size | $\mathscr{B} = 2^{10}$ |
| HE-plain slot size | $S = 32$ |

data as the benchmark to analyze the effectiveness of the detection model after applying the proposed PPDiL.

### Evaluation Metrics

To evaluate the performance of the detection model, the confusion matrix is utilized, which is suitable for a machine learning-based classification system [73]. We denote TP, TN, FP, and FN as, respectively, "True Positive", "True Negative", "False Positive", and "False Negative". Assuming the system consists of $C$ classes, which include normal and attack traffic, the accuracy can be calculated as:

$$Accuracy = \frac{1}{C}\sum_{c=1}^{C}\frac{\mathrm{TP}_c + \mathrm{TN}_c}{\mathrm{TP}_c + \mathrm{TN}_c + \mathrm{FP}_c + \mathrm{FN}_c}. \tag{3.21}$$

The macro-average precision and recall are utilized in this term. The macro-average precision is:

$$Precision = \frac{1}{C}\sum_{c=1}^{C}\frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c}. \tag{3.22}$$

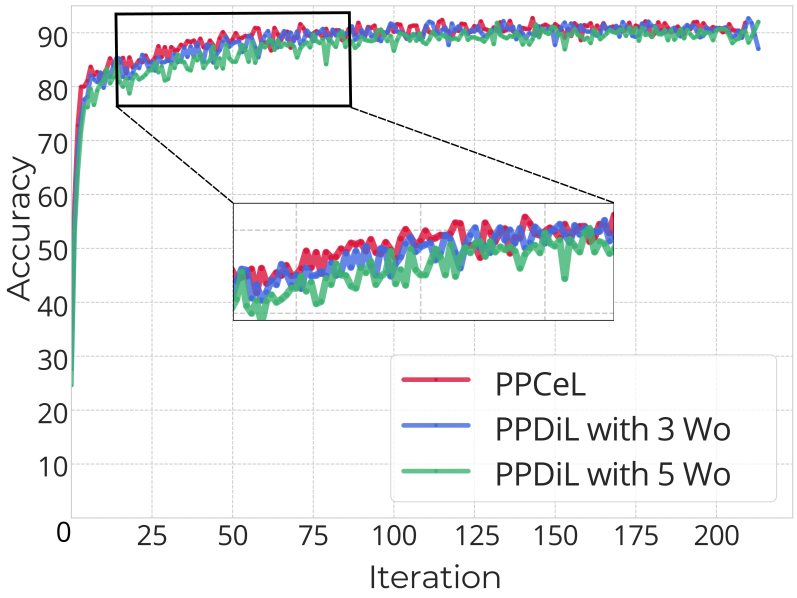



Fig. 3.6 Convergence of the considered learning algorithms.

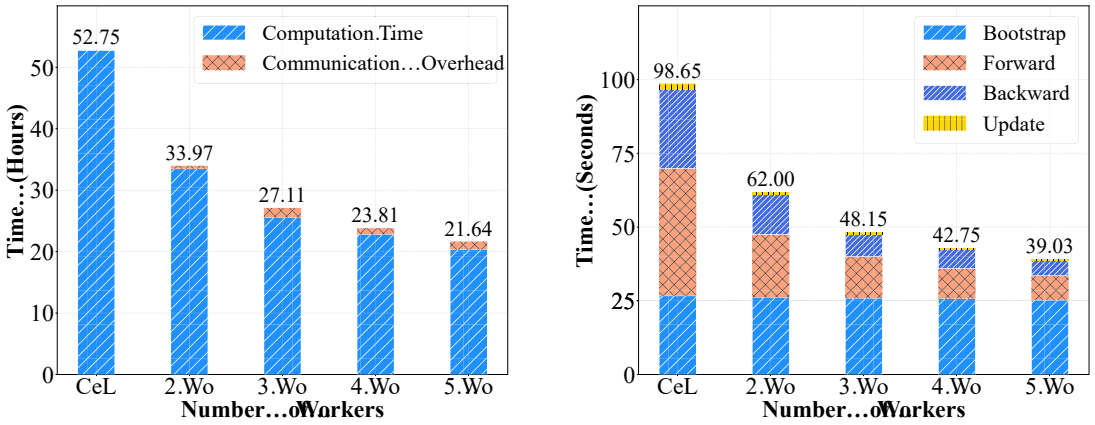
The macro-average recall is calculated as follows:

$$Recall = \frac{1}{C}\sum_{c=1}^{C}\frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FN}_c}. \tag{3.23}$$

### 3.3.2 Simulation Results

#### Convergence Analysis

In this subsection, we examine the convergence rate of our proposed PPDiL. Fig. 3.6 demonstrates the HE-encrypted training process of the centralized approach (PPCeL) and our proposed distributed approach (PPDiL), where the scenario of three workers and five workers are considered. Accordingly, the PPCeL reaches convergence after around 75 iterations due to the processing of a large dataset at a centralized point. In contrast, the PPDiL exhibits a slower convergence rate than PPCeL, with the convergence speed decreasing as more workers join the process. In detail, the PPDiL with five workers requires nearly 90 iterations to achieve stable accuracy, while the PPDiL with three workers is nearly equivalent to the PPCeL, reaching convergence after 75 iterations. However, despite the gap in convergence speed, the accuracy of PPDiL and PPCeL remains consistent, regardless of whether the number of workers increases. To be more specific, both the accuracy of PPCeL and PPDiL




(a) Total runtime during training process

(b) Training time in one iteration

Fig. 3.7 Execution time of offline training phase.

remains the same after 200 iterations, stabilizing at around 91%. As a result, moving from PPCeL to PPDiL training can still achieve effective accuracy during the learning process.

### Training Time Evaluation

As described in Fig. 3.7, we comprehensively analyze the improvement of our proposed PPDiL compared to the CeL approach. In Fig. 3.7(a), we first provide the total training time of CeL and PPDiL in which distributing the training tasks to multiple workers shows an improvement in processing time. While the CeL requires 52.75 hours to finish the training, the proposed PPDiL can decrease the computation time significantly to 33.97, 27.11, 23.81, and 21.64 hours by distributing the learning tasks to two workers, three workers, four workers, and five workers, respectively. However, dividing the learning tasks among multiple workers leads to a delay in connection, which is described as the communication overhead. As can be seen from Fig. 3.7(a), the PPDiL with two workers has low communication overhead because the PPDiL only need to maintain the connection between a given master node and two worker nodes. Meanwhile, scenarios involving more workers incur larger communication overhead due to the varying hardware capabilities of the different participants. This discrepancy results in faster workers having to wait for slower ones to complete their rounds, leading to delays in the synchronization time during the training phase.

Additionally, Fig. 3.7(b) provide a comprehensive analysis of the computation time over different stages during training (i.e., forward, backward, update, and bootstrap) in one training iteration. As the bootstrap mechanism is only applied to the neural networks' layers, its processing time remains consistent across CeL and other PPDiL scenarios, averaging

around 25 seconds. For other training tasks, the PPDiL demonstrates a significant advantage by linearly reducing computation time, from 98.65 seconds with CeL down to 62 seconds with 2 workers and 39.03 seconds with 5 workers. It is worth noting that aside from the bootstrap time, the processing time for other tasks in PPDiL decreases exponentially as the number of distributed workers increases, particularly in comparison to CeL.

To be more specific, Table 3.2 illustrates a more detailed analysis of Fig. 3.7(b) regarding computational over each layer. Considering the forward process of CeL and PPDiL with 2 workers, the computation times for CeL over the input layer, hidden layer, and output layer are 10.376 seconds, 5.145 seconds, and 8.074 seconds, respectively. In contrast, the forward process time for PPDiL with 2 workers over the corresponding neural network layers is significantly reduced to 5.14 seconds, 2.478 seconds, and 3.885 seconds, respectively, which represent half of the computation times for CeL. Following the same trend of PPDiL with 2 workers, the 5-worker scenario illustrates one-fifth of computation time compared with CeL, which respectively remains at 1.923 seconds, 0.936 seconds, and 1.485 seconds in the considered layers. However, the bootstrapping during the update process remains identical to both CeL and DL (i.e., with 2 to 5 workers) approaches, maintained from around 8 to 9 seconds. Therefore, this leads to a non-exponential decrease in training time. As a result, based on our real implementation of the proposed PPDiL framework, despite the impact of the bootstrapping time and the communication overhead, the training time is still improved. The participation of additional workers in the training process leads to significantly faster learning time compared to the centralized learning approach.

### Cyberattack Detection Ability Evaluation

In this subsection, we further demonstrate the accuracy of the proposed PPDiL with the classification results of non-encrypted and encrypted data. Table 3.3 compares the non-encrypted data detection performance of the normal training and the privacy-preserving training regarding both centralized and distributed approaches. In general, the accuracy, precision, and recall of the learning model trained with non-encrypted data (i.e., CeL and DiL) and the ones trained with encrypted data (i.e., PPCel and PPDiL) remain nearly identical. Due to the centralized approach, PPCeL achieves an accuracy of 91.45%, which is only marginally lower than CeL's 91.487%, with a negligible difference of approximately 0.04%. This slight gap underscores that both methods consistently deliver around 91.4% accuracy, demonstrating the effectiveness of HE-encrypted training. A similar trend is observed regarding our proposed distributed approaches, where the results are consistent in different scenarios compared to non-encrypted training benchmarks. Notably, the results of PPDiL maintain the same accuracy level as DiL, with the gap nearly from 0.01 to 0.02%. For

Table 3.2 Processing time over one training iteration

| **Process** | **Layer** | **Time (seconds)** | | | | |
|---|---|---|---|---|---|---|
| | | **CeL** | **2 Wo** | **3 Wo** | **4 Wo** | **5 Wo** |
| Forward | Input Layer | 10.376 | 5.140 | 3.394 | 2.416 | 1.923 |
| | SiLU 1 | 6.954 | 3.487 | 2.223 | 1.698 | 1.455 |
| | Hidden Layer | 5.145 | 2.478 | 1.643 | 1.174 | 0.936 |
| | SiLU 2 | 8.451 | 4.222 | 2.752 | 1.965 | 1.561 |
| | Output Layer | 8.074 | 3.885 | 2.695 | 1.852 | 1.485 |
| | SiLU 3 | 4.164 | 2.102 | 1.357 | 1.062 | 0.867 |
| Backward | Output Layer | 6.993 | 3.521 | 2.228 | 1.726 | 1.384 |
| | Hidden Layer | 12.325 | 6.209 | 4.060 | 2.845 | 2.329 |
| | Input Layer | 7.497 | 3.713 | 2.417 | 1.876 | 1.526 |
| Update | Input Layer | 0.695 | 0.324 | 0.171 | 0.116 | 0.095 |
| | Bootstrap 1 | 8.874 | 8.591 | 8.623 | 8.547 | 8.472 |
| | Hidden Layer | 0.727 | 0.394 | 0.201 | 0.150 | 0.125 |
| | Bootstrap 2 | 8.778 | 8.713 | 8.619 | 8.676 | 8.429 |
| | Output Layer | 0.589 | 0.341 | 0.189 | 0.145 | 0.119 |
| | Bootstrap 3 | 9.008 | 8.877 | 8.637 | 8.502 | 8.327 |

Table 3.3 Performance comparisons of CeL/DiL and PPCeL/PPDiL with non-encrypted detection

| **Model** | **Centralized** | | **Distributed** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **CeL** | **PPCeL** | **2 Wo** | | **3 Wo** | | **4 Wo** | | **5 Wo** | |
| | | | **DiL** | **PPDiL** | **DiL** | **PPDiL** | **DiL** | **PPDiL** | **DiL** | **PPDiL** |
| **Accuracy** | 91.487 | **91.450** | 91.483 | **91.375** | 91.475 | **91.350** | 91.016 | **90.875** | 91.024 | **90.925** |
| **Precision** | 91.796 | **91.749** | 91.921 | **92.092** | 92.071 | **91.610** | 91.505 | **91.343** | 91.431 | **91.433** |
| **Recall** | 91.507 | **91.465** | 91.441 | **91.339** | 91.476 | **91.435** | 91.036 | **90.941** | 91.086 | **90.931** |

instance, in terms of distributed learning with three workers, the DiL achieves an accuracy of 91.475%, which is slightly higher than the accuracy of 91.35% from PPDiL. As a result, this minor gap is acceptable, demonstrating the reliability of our proposed PPDiL. Additionally, the results from Table 3.3 also indicate the comparison between centralized and distributed approaches. We can observe that the PPDiL achieves an accuracy level nearly identical to PPCeL, in which the gap across different scenarios (i.e., two Wo to five Wo) ranges from approximately 0.1 to 0.5. However, the classification results in Fig. 3.8 show that the detection performance of PPDiL for each attack type with five workers is still accurate compared to the PPCeL.

Table 3.4 Detection result of PPDiL with encrypted data

| **Model** | **PPCeL** | **PPDiL** | | | |
|---|---|---|---|---|---|
| | | **2 Wo** | **3 Wo** | **4 Wo** | **5 Wo** |
| **Accuracy** | 91.725 | 91.450 | 90.301 | 90.875 | 90.805 |
| **Precision** | 92.002 | 92.293 | 91.550 | 91.369 | 91.337 |
| **Recall** | 91.729 | 91.382 | 91.378 | 90.944 | 90.828 |

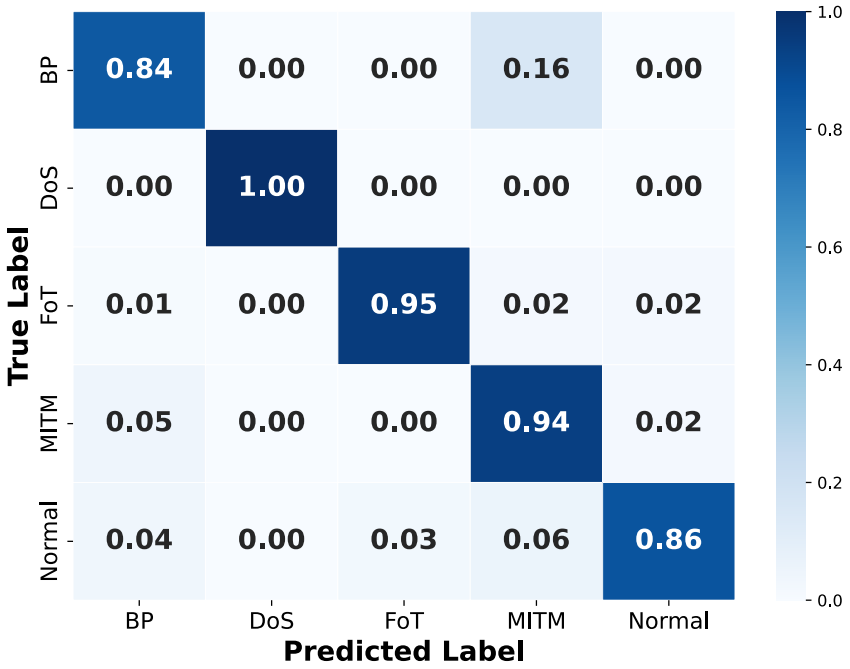
Moreover, Table 3.4 shows the detection performance of PPDiL over the encrypted data. As can be seen, the accuracy of the encrypted detection is nearly the same as its non-encrypted counterparts, remaining around 91% in different scenarios. Hence, the trivial gap in the aforementioned results is due to the HE evaluation, as the noise is added during the homomorphic computation over encrypted data. Despite the added noise during HE computation, the detection results of our learning model still consistently maintain between 91% and 92%. Consequently, our proposed method enables accurate detection for both raw data and encrypted data, offering the versatility needed for real-world applications where encrypted input classification is required.

### 3.3.3 Blockchain Nodes-enabled HE Computational Evaluation

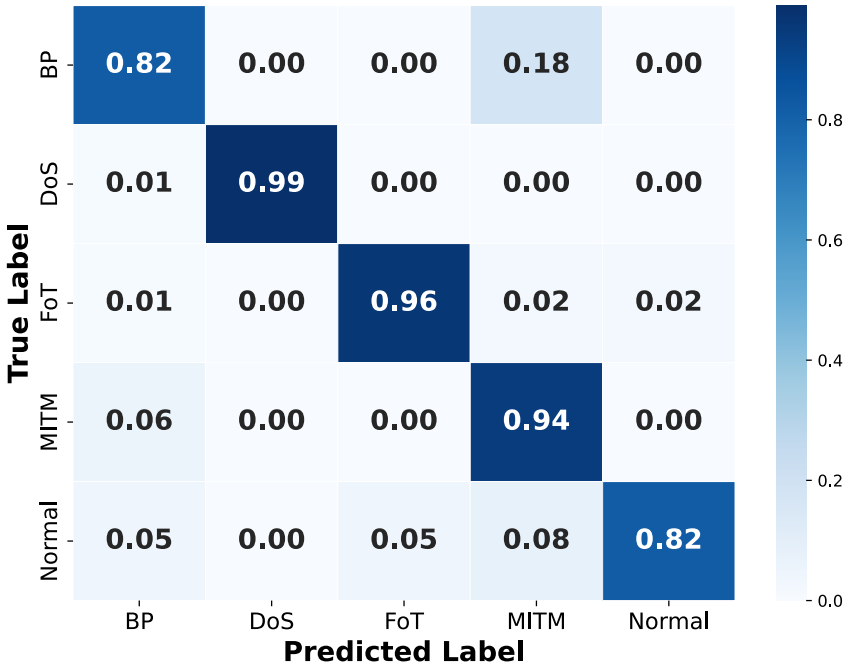
#### Experiment Setup

To evaluate the reliability of our proposed cyberattack detection framework, we perform the experiment on a blockchain node (BN) with multiple consensus mechanisms regarding the network data encryption process while mining the IoT data. In the experiment, we consider the different batch sizes of the local network data, including 50, 500, 1000, and 2000 samples. The implementation consists of a blockchain node and an IoT transaction issuer in which the configuration of the devices and software for the implementation is described as follows:

- The BN operates on processor Intel® Core i9-13900K (24 cores, 32 threads). The IoT transaction issuer is run on the processor Intel® Core i7 with 16GB RAM.
- The BN with PoW and PoA mechanisms is launched by Go-Ethereum v1.10.14 (*Geth*) [78] - an official open-source implementation of the Ethereum protocol.
- The BN with PoS mechanism is launched by *Geth* v1.14.6 and *Prysm* v5.0.3 [79] - an official implementation of the PoS consensus mechanism in Ethereum 2.0. It is worth noting that in the PoS node, *Geth* operates as the chain execution and *Prysm* provides the PoS configuration as a third-party software.

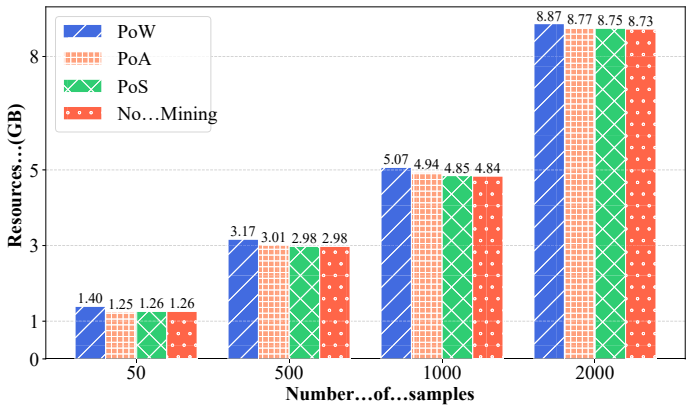



(a) Confusion matrix of PPCeL

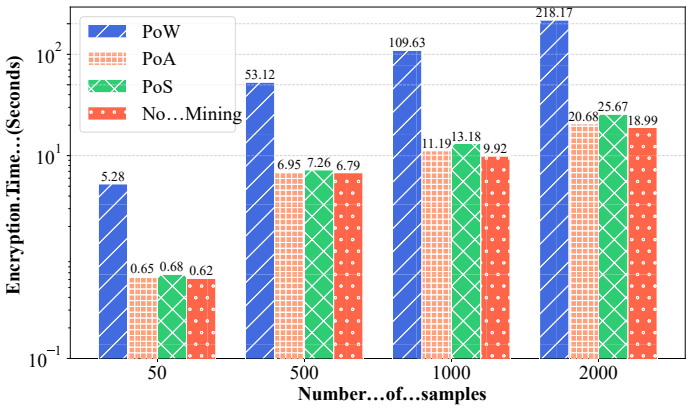



(b) Confusion matrix of PPDiL with five workers

Fig. 3.8 Classification results of the non-encrypted detection within the proposed HE-encrypted deep neural network.




(a) Computing resources of encryption with different batches size




(b) Latency of encryption within different batches size

Fig. 3.9 Evaluation of the *Geth* blockchain node-enabled HE in different consensus mechanisms: PoW, PoA, PoS, and no mining.

### Experiment Results

In Fig. 3.9, we examine our proposed framework with different mining algorithms in terms of computational resources and the latency time during the encryption process. As illustrated in Fig. 3.9(a), the PoW blockchain node slightly costs higher resources during the encryption, with around 0.15 GB higher than other counterparts. Despite varying batch sizes, the encryption processes for PoA and PoS nodes consistently require computing resources nearly identical to those in scenarios without mining. Known for their lightweight nature, PoA and PoS maintain stable resource usage for encryption across 50, 500, 1000, and 2000 samples, holding steady at approximately 1.26 GB, 2.98 GB, 4.85 GB, and 8.75 GB, respectively. As a result, the mining process does not affect the resources of the BN during the HE-based encryption.

To further demonstrate the scalability of our proposed framework, we analyze the latency time of the encryption during various scenarios. As observed in Fig. 3.9(b), due to the intensive computational demands of PoW, the encryption process of the PoW-based BN incurs the highest latency, spanning from 5.28 seconds for 50 samples to 218.17 seconds for 2,000 samples. This results in latency that is approximately 8 to 12 times greater than that of a node operating without a mining process. In contrast to the trend in resources analysis, PoA and PoS incur slightly longer encryption times compared to nodes without a mining mechanism. In particular, the PoS node experiences delays ranging from 0.07 to nearly 7 seconds, primarily due to the reliance on third-party software to maintain the PoS algorithm, which reduces the resources available for data encryption. On the other hand, although PoA maintains lower latency than PoS, it still experiences slightly higher latency than the no-mining scenario, with minimal delays ranging from 0.06 to 1.69 seconds across different batch sizes. Consequently, while HE latency does increase across the three mining mechanisms, the slight latency difference in the lightweight algorithms (i.e., PoS and PoA) is minimal, suggesting that our proposed framework remains a viable option for various blockchain-based IoT networks.

### 3.3.4 Encrypted Inference Time Analysis

#### Hardware Configuration

In this section, we perform the experiment to examine the adaptability of real-time detection with incoming encrypted data within our framework. During the implementation, we deploy the trained model, optimized through the proposed PPDiL, to detect the encrypted samples across various hardware configurations, which are described as follows:

- Device-1: Intel Xeon E-2288G @3.7GHz processors with 8 processor cores.
- Device-2: Intel Xeon Gold 6238R @2.2GHz processors with 28cores (26 cores enabled).
- Device-3: AMD EPYC 9354P 3.25GHz processors with 32 cores 256MB L3 Cache.

Similar to the approach in [61], we consider the incoming network traffic data to be extracted in data frames, with each containing 400 samples for our experiment.

#### Experiment Results

To analyze the effectiveness of our proposed cyberattack detection framework, we consider the processing time of the detection with encrypted samples. As observed in Fig. 3.10, the

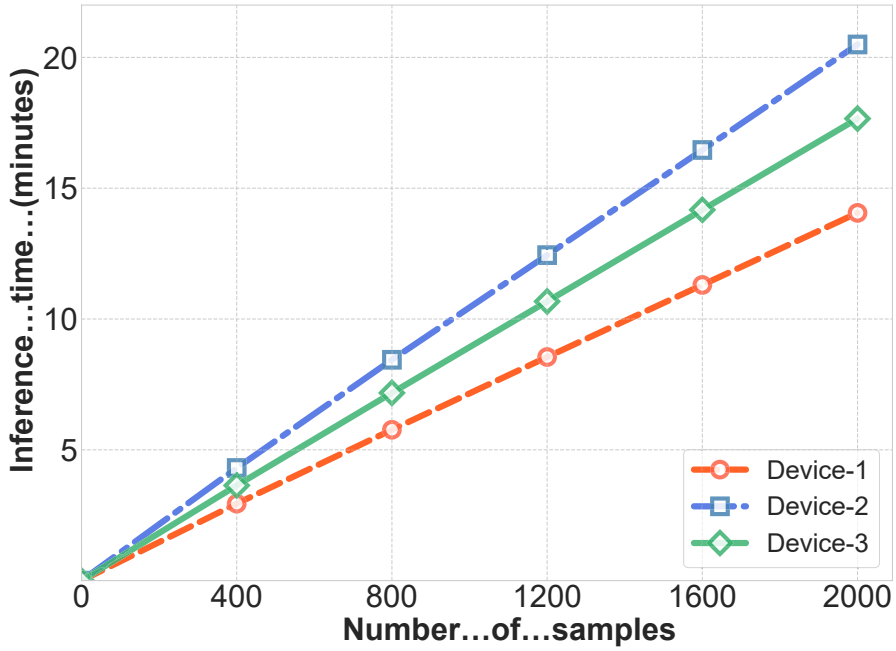



Fig. 3.10 The inference time of the detection model with encrypted samples.

trained model employing our proposed PPDiL requires a maximum of nearly 5 minutes to predict the considered encrypted data frame with the deployment in Device-2 and Device-3. Meanwhile, regarding the utilization of Device-1 with more powerful hardware configurations, the inference time of the trained detection model shows significant improvement. Here, the trained model deployed on Device-1 can effectively detect the 400 encrypted samples in around 2.5 minutes, resulting in a throughput of approximately 2.4 samples per second. Therefore, the analysis indicates that our proposed cyberattack detection framework can be effectively adapted to real-world systems, especially with the enhanced computing power offered by modern edge servers[1].

## 3.4 Summary

In this chapter, we have proposed a novel privacy-preserving cyberattack detection framework for IoT-based blockchain networks, addressing computational challenges by allowing blockchain nodes to share IoT data with a cloud service provider (CSP) for training machine learning models. The framework leverages our proposed SIMD packing algorithms and

[1]https://aws.amazon.com/edge/

CKKS encryption scheme to securely process and transmit data, enabling encrypted training on the CSP. We have introduced two complementary learning methods, including a deep neural network training algorithm and a distributed learning algorithm, both designed for efficiency and privacy. Our simulations and experiments have demonstrated that the proposed approach achieves 91% accuracy with minimal processing time, closely matching the non-encrypted baseline, proving its potential for real-world application.

# Chapter 4

# Conclusion and Future Research Directions

## 4.1 Conclusion

In this thesis, we have presented our contributions to safeguarding both security and privacy in IoT systems by effectively combining deep learning with homomorphic encryption. The thesis provides a comprehensive exploration of privacy-preserving deep learning for cyberattack detection across diverse IoT applications, focusing on two key domains: (i) Internet of Vehicles and (ii) Blockchain-based Internet of Things. The key findings from the research works can be summarized as follows. Firstly, the network data from IoT participants can be analyzed by machine learning (ML) and deep learning (DL) models to efficiently detect incoming cyberattacks, thus enabling IoT systems to proactively counter potential threats, including zero-day attacks. Secondly, applying ML/DL models in IoT systems leads to privacy issues since IoT users' private data is readable by the models' owners (e.g., cloud providers). To preserve data privacy, homomorphic encryption (HE) can be leveraged to integrate with learning models. This cryptographic method allows direct computations on ciphertext without needing to decrypt, thereby maintaining data confidentiality during the learning process.

In the first study, detailed in Chapter 2, we introduced an innovative privacy-preserving federated learning (FL) framework tailored for intrusion detection in the Internet of Vehicles (IoVs) with constrained computational resources. Our framework addresses the challenge of local training on resource-limited vehicles by enabling them to offload data to a centralized server. To safeguard user privacy, homomorphic encryption (HE) is applied to the data prior to transmission. This encryption allows the server to process the data using our

proposed HE-based training algorithm, enabling it to learn from the encrypted information without accessing its raw content. This ensures user privacy throughout the learning process while paving the way for efficient FL deployment in practical IoV networks. Simulation results demonstrate that the proposed framework effectively detects cyberattacks in IoV environments. While the accuracy of the encrypted neural network is marginally lower than that of its unencrypted counterpart, the trade-off is acceptable and presents opportunities for further optimization in future research.

In the second study, presented in Chapter 3, we proposed an advanced privacy-preserving framework for detecting cyberattacks in IoT-based blockchain networks. This framework addresses the computational limitations of blockchain nodes by enabling them to offload IoT data to a cloud service provider (CSP) for machine learning model training. To ensure secure data transmission and processing, we integrated our novel SIMD packing algorithms with the CKKS encryption scheme, facilitating encrypted training directly on the CSP. The framework introduces two complementary learning approaches: a deep neural network training algorithm and a distributed learning algorithm, both optimized for efficiency and privacy. Simulation results and experiments reveal that the proposed solution achieves a remarkable 91% accuracy with minimal processing overhead, closely aligning with the performance of unencrypted baselines. These results underscore the framework's practicality and effectiveness for real-world applications.

## 4.2 Future Research Directions

### 4.2.1 Current Research Limitations

While our proposed approach demonstrates promising results, several challenges should be addressed. Below, we outline key challenges that could guide future improvements and refinements:

- Given the decentralized nature of blockchain networks, collecting real-world attack samples is challenging. To further strengthen the adaptability of AI-driven security solutions, future research should not only expand the scope of attack types and datasets but also explore more advanced learning approaches that enable models to detect emerging threats with minimal labelled data. This could enhance the model's ability to generalize across diverse attack scenarios while maintaining high detection accuracy.

- Collecting attack data across different blockchain consensus mechanisms is both challenging and crucial, as each mechanism introduces unique vulnerabilities and

attack vectors. Currently, the study in Chapter 3 focuses only on attacks targeting PoW-based blockchain networks. To improve adaptability, it is essential to assess the stability of different consensus mechanisms under various attack scenarios.

### 4.2.2 Future Directions

Future research works can potentially consider improving not only the adaptability and scalability but also the HE-enabled privacy-preserving ML framework with more DL approaches, e.g., convolutional neural network, transformer, and advanced federated learning. Particularly, various research directions can be considered:

- The combination of HE with IoT applications can be potentially explored through a multi-layer edge computing architecture, where computational tasks are dynamically distributed across cloud and edge nodes. With the ongoing advancements in edge devices and the initial integration of HE with GPU acceleration, a more effective balance between security and computational efficiency can be achieved by leveraging shared computing capabilities between cloud infrastructure and IoT edge nodes. Nevertheless, a major challenge is ensuring low-latency processing while maintaining security in a multi-layer edge computing architecture, which can be addressed by optimizing HE schemes and employing adaptive workload distribution strategies.

- The scalability of our approach can be further explored by extending experiments to a large-scale private blockchain network or an IoT environment with multiple interconnected nodes. Investigating the transmission of different amounts of encrypted data to the server in real-time would provide deeper insights into system efficiency under dynamic conditions. Furthermore, the development of an application demo would enable the optimization of critical performance metrics such as throughput and communication efficiency, paving the way for practical deployment in next-generation secure systems. Although managing large-scale encrypted data transmission can cause communication bottlenecks and latency, efficient encrypted compression techniques and communication-aware scheduling can potentially solve the problem.

- The integration of knowledge distillation (KD) with the proposed privacy-preserving FL framework. This approach would allow both the generalization feature and personalization to the FL system, enabling the processing of non-independent and identically distributed (non-iid) data of heterogeneous users. By combining with KD, the framework can transfer knowledge from a global model to local models effectively, enhancing the system's ability to handle diverse user-specific data distributions while maintaining

robust performance and data privacy. However, a major challenge is ensuring that knowledge transfer does not degrade model performance due to the limitations of encrypted training data. A potential solution is to design adaptive KD mechanisms that can effectively distil useful representations while preserving privacy.

- The potential of HE-enabled FL can be explored across a range of FL architectures. For instance, the development of multi-modal FL with HE-encrypted data offers a pathway to privacy-preserving ML that processes data spanning multiple layers of communication and networking systems. This approach significantly reduces the risk of data leakage, thereby enhancing the security and privacy of next-generation wireless systems. Nevertheless, the high computational cost of HE in FL training could hinder real-time processing. In this regard, leveraging model quantization techniques can help to optimize performance without compromising privacy.

- Future research could investigate the potential of the proposed FL framework within zero-trust networking environments. By leveraging the training algorithm with encrypted data, privacy-preserving anomaly detection algorithms can be developed, ensuring users can actively participate in a zero-trust network without compromising their sensitive data. This approach not only enhances security and privacy but also fosters trust in zero-trust systems where users' data is continuously managed and processed. A key challenge is the computational overhead of encrypted anomaly detection, which can be mitigated by optimizing hybrid encryption approaches and using lightweight/tiny AI detection models.

- Enhancing the integration of homomorphic encryption (HE) with deep learning (DL) can be achieved by developing advanced training and inference algorithms using state-of-the-art DL models. For example, implementing the training of convolutional neural networks (CNNs) or recurrent neural networks (RNNs) on HE-encrypted data with reduced processing time and high throughput holds significant potential. This approach not only advances the field of privacy-preserving machine learning for cyberattack detection but also opens new possibilities for its application in broader domains of artificial intelligence. A key challenge lies in the inefficiency of HE when handling deep models, especially in scenarios requiring high throughput, which can lead to significant computational overhead and increased latency. Employing polynomial approximations of non-linear functions and optimizing encrypted matrix operations to improve inference throughput could be a promising solution in this direction.

# References


[1] K. Han, S. Hong, J. H. Cheon, and D. Park, "Efficient logistic regression on large encrypted data," Cryptology ePrint Archive, 2018. [Online]. Available: https://eprint.iacr.org/2018/662

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] L. S. Vailshery, "Number of iot connections worldwide 2022-2033, with forecasts to 2030," https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[4] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[5] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali, and M. Guizani, "A survey of machine and deep learning methods for internet of things (iot) security," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1646–1685, 2020.

[6] M. Antonakakis, T. April, M. Bailey *et al.*, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017, pp. 1093–1110.

[7] Zscaler, "Anatomy of a cloud breach: How 100 million credit card numbers were exposed." https://www.zscaler.com/resources/white-papers/capital-one-data-breach.pdf/.

[8] G. R. David Atch and R. Bevington, "Microsoft defender: How to proactively defend against mozi iot botnet," https://www.microsoft.com/en-us/security/blog/2021/08/19/how-to-proactively-defend-against-mozi-iot-botnet.

[9] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, and M. H. Rehmani, "Applications of blockchains in the internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1676–1717, 2019.

[10] S. Nakatomo, "Bitcoin: A peer-to-peer electronic cash system." [Online]. Available: https://bitcoin.org/bitcoin.pdf

[11] IBM, "What is IoT with blockchain?" Accessed: August. 13, 2024. [Online]. Available: https://www.ibm.com/topics/blockchain-iot

[12] IOTA, "Enabling privacy and control of healthcare data," Accessed: August. 13, 2024. [Online]. Available: https://www.iota.org/solutions/ehealth

[13] Slowmist, "Slowmist hacked statistical," Accessed: July. 16, 2024. [Online]. Available: https://hacked.slowmist.io/statistics/?c=all&d=all

[14] TechMonitor, "The biggest cryptocurrency hacks of all time," Accessed: July. 16, 2024. [Online]. Available: https://techmonitor.ai/technology/cybersecurity/biggest-cryptocurrency-hacks-of-all-time

[15] NewsDesk, "Btc turk hack: Exchange reports cyber attack with €51 million stolen," Accessed: July. 16, 2024. [Online]. Available: https://www.unlock-bc.com/124941/btc-turk-hack-exchange-reports-cyber-attack/

[16] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. De Alvarenga, "A survey of intrusion detection in internet of things," *Journal of Network and Computer Applications*, vol. 84, pp. 25–37, 2017.

[17] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, "Network intrusion detection for iot security based on learning techniques," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2671–2701, 2019.

[18] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali, and M. Guizani, "A survey of machine and deep learning methods for internet of things (IoT) security," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1646–1685, 2020.

[19] M. Ul Hassan, M. H. Rehmani, and J. Chen, "Anomaly detection in blockchain networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 289–318, 2023.

[20] M. Keshk, E. Sitnikova, N. Moustafa, J. Hu, and I. Khalil, "An integrated framework for privacy-preserving based anomaly detection for cyber-physical systems," *IEEE Transactions on Sustainable Computing*, vol. 6, no. 1, pp. 66–79, 2021.

[21] B. Liu, M. Ding, S. Shaham, W. Rahayu, F. Farokhi, and Z. Lin, "When machine learning meets privacy: A survey and outlook," *ACM Computing Survey*, vol. 54, no. 2, 2021.

[22] S. Hui, Z. Wang, X. Hou, X. Wang, H. Wang, Y. Li, and D. Jin, "Systematically quantifying IoT privacy leakage in mobile networks," *IEEE Internet of Things Journal*, vol. 8, no. 9, pp. 7115–7125, 2021.

[23] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[24] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang, "Secure logistic regression based on homomorphic encryption: Design and evaluation," *JMIR Med Inform*, vol. 6, no. 2, p. e19, Apr. 2018.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[26] D. T. Hoang, N. Van Huynh, D. N. Nguyen, E. Hossain, and D. Niyato, *Deep Reinforcement Learning for Wireless Communications and Networking: Theory, Applications and Implementation*. John Wiley & Sons, 2023.

[27] J. Li, T. Tang, W. X. Zhao, J.-Y. Nie, and J.-R. Wen, "Pre-trained language models for text generation: A survey," *ACM Comput. Surv.*, vol. 56, no. 9, 2024.

[28] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, p. 139–144, 2020.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[31] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[32] 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*, vol. 54. PMLR, Apr. 2017, pp. 1273–1282.

[33] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[34] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*. Springer, 2006, pp. 265–284.

[35] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT*, Switzerland, Nov. 2017, pp. 409–437.

[36] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Advances in Cryptology – CRYPTO 2012*. Springer Berlin Heidelberg, 2012, pp. 868–886.

[37] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. Association for Computing Machinery, 2012, p. 309–325.

[38] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[39] B. D. Manh, C.-H. Nguyen, D. T. Hoang, and D. N. Nguyen, "Homomorphic encryption-enabled federated learning for privacy-preserving intrusion detection in resource-constrained iov networks," in *2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall)*, 2024, pp. 1–6.

[40] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot—network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.

[41] N. Moustafa, "A new distributed architecture for evaluating ai-based security systems at the edge: Network Ton_IoT datasets," *Sustainable Cities and Society*, vol. 72, p. 102994, 2021.

[42] P. V. Dinh, Q. U. Nguyen, D. T. Hoang, D. N. Nguyen, S. P. Bao, and E. Dutkiewicz, "Constrained twin variational auto-encoder for intrusion detection in iot systems," *IEEE Internet of Things Journal*, vol. 11, no. 8, pp. 14 789–14 803, 2024.

[43] A. Zekry, A. Sayed, M. Moussa, and M. Elhabiby, "Anomaly detection using iot sensor-assisted ConvLSTM models for connected vehicles," in *IEEE 93rd Vehicular Technology Conference*, 2021, pp. 1–6.

[44] T. Alladi, V. Kohli, V. Chamola, F. R. Yu, and M. Guizani, "Artificial intelligence (AI)-empowered intrusion detection architecture for the internet of vehicles," *IEEE Wireless Communications*, vol. 28, no. 3, pp. 144–149, 2021.

[45] T. V. Khoa, Y. M. Saputra, D. T. Hoang, N. L. Trung, D. Nguyen, N. V. Ha, and E. Dutkiewicz, "Collaborative learning model for cyberattack detection systems in iot industry 4.0," in *IEEE Wireless Communications and Networking Conference*, 2020, pp. 1–6.

[46] T. V. Khoa, D. T. Hoang, N. L. Trung, C. T. Nguyen, T. T. T. Quynh, D. N. Nguyen, N. V. Ha, and E. Dutkiewicz, "Deep transfer learning: A novel collaborative learning model for cyberattack detection systems in iot networks," *IEEE Internet of Things Journal*, vol. 10, no. 10, pp. 8578–8589, 2023.

[47] H. Liu, S. Zhang, P. Zhang, X. Zhou, X. Shao, G. Pu, and Y. Zhang, "Blockchain and federated learning for collaborative intrusion detection in vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 6073–6084, 2021.

[48] B. Ji, X. Zhang, S. Mumtaz, C. Han, C. Li, H. Wen, and D. Wang, "Survey on the internet of vehicles: Network architectures and applications," *IEEE Communications Standards Magazine*, vol. 4, no. 1, pp. 34–41, 2020.

[49] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[50] N. Yoshida, T. Nishio, M. Morikura, K. Yamamoto, and R. Yonetani, "Hybrid-fl for wireless networks: Cooperative learning mechanism using non-iid data," in *IEEE International Conference on Communications*, 2020, pp. 1–7.

[51] Z. Ji, L. Chen, N. Zhao, Y. Chen, G. Wei, and F. R. Yu, "Computation offloading for edge-assisted federated learning," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 9330–9344, 2021.

[52] C.-H. Nguyen, B. D. Manh, D. Thai Hoang, D. N. Nguyen, and E. Dutkiewicz, "Towards secure ai-empowered vehicular networks: A federated learning approach using homomorphic encryption," in *2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall)*, 2024, pp. 1–6.

[53] J. Otávio Chervinski, D. Kreutz, and J. Yu, "Analysis of transaction flooding attacks against monero," in *2021 IEEE International Conference on Blockchain and Cryptocurrency*, 2021, pp. 1–8.

[54] X. Wang, X. Zha, G. Yu, W. Ni, R. P. Liu, Y. J. Guo, X. Niu, and K. Zheng, "Attack and defence of ethereum remote apis," in *2018 IEEE Globecom Workshops*, 2018, pp. 1–6.

[55] B. D. Manh, C.-H. Nguyen, D. T. Hoang, D. N. Nguyen, M. Zeng, and Q.-V. Pham, "Privacy-preserving cyberattack detection in blockchain-based iot systems using ai and homomorphic encryption," *IEEE Internet of Things Journal*, pp. 1–1, 2025.

[56] V. T. Truong and L. B. Le, "Metacids: Privacy-preserving collaborative intrusion detection for metaverse based on blockchain and online federated learning," *IEEE Open Journal of the Computer Society*, vol. 4, pp. 253–266, 2023.

[57] A. Aljuhani, P. Kumar, R. Alanazi, T. Albalawi, O. Taouali, A. K. M. N. Islam, N. Kumar, and M. Alazab, "A deep-learning-integrated blockchain framework for securing industrial iot," *IEEE Internet of Things Journal*, vol. 11, no. 5, pp. 7817–7827, 2024.

[58] J. Kim, M. Nakashima, W. Fan, S. Wuthier, X. Zhou, I. Kim, and S.-Y. Chang, "Anomaly detection based on traffic monitoring for secure blockchain networking," in *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2021, pp. 1–9.

[59] W. Cao, Y. Huang, D. Li, F. Yang, X. Jiang, and J. Yang, "A blockchain based link-flooding attack detection scheme," in *IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 4, 2021, pp. 1665–1669.

[60] D. H. Son, B. D. Manh, T. V. Khoa, N. L. Trung, D. T. Hoang, H. T. Minh, Y. Alem, and L. Q. Minh, "Semi-supervised learning for anomaly detection in blockchain-based supply chains," *arXiv preprint arXiv:2407.15603*, 2024.

[61] T. V. Khoa, D. H. Son, D. T. Hoang, N. L. Trung, T. T. T. Quynh, D. N. Nguyen, N. V. Ha, and E. Dutkiewicz, "Collaborative learning for cyberattack detection in blockchain networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 54, no. 7, pp. 3920–3933, 2024.

[62] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 48. PMLR, 20–22 Jun 2016, pp. 201–210.

[63] S. Meftah, B. H. M. Tan, C. F. Mun, K. M. M. Aung, B. Veeravalli, and V. Chandrasekhar, "Doren: Toward efficient deep convolutional neural networks with fully homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3740–3752, 2021.

[64] M. Kim, X. Jiang, K. Lauter, E. Ismayilzada, and S. Shams, "Secure human action recognition by encrypted neural network inference," *Nature Communications*, vol. 13, no. 1, p. 4799, 2022.

[65] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *Proceedings on Privacy Enhancing Technologies*, no. 3, pp. 123–142, 2018.

[66] K. Nandakumar, N. Ratha, S. Pankanti, and S. Halevi, "Towards deep neural network training on encrypted data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[67] C.-H. Nguyen, Y. M. Saputra, D. T. Hoang, D. N. Nguyen, V.-D. Nguyen, Y. Xiao, and E. Dutkiewicz, "Encrypted data caching and learning framework for robust federated learning-based mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2705–2720, 2024.

[68] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30 039–30 054, 2022.

[69] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Networks*, vol. 107, pp. 3–11, 2018.

[70] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-IIoTset: A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning," *IEEE Access*, vol. 10, pp. 40 281–40 306, 2022.

[71] X. Zhang, L. Hao, G. Gui, Y. Wang, B. Adebisi, and H. Sari, "An automatic and efficient malware traffic classification method for secure internet of things," *IEEE Internet of Things Journal*, vol. 11, no. 5, pp. 8448–8458, 2024.

[72] H. Jahangir, S. Lakshminarayana, C. Maple, and G. Epiphaniou, "A deep-learning-based solution for securing the power grid against load altering threats by iot-enabled devices," *IEEE Internet of Things Journal*, vol. 10, no. 12, pp. 10 687–10 697, 2023.

[73] T. Fawcett, "An Introduction to ROC Analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, June. 2006.

[74] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, "Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities," *IEEE Access*, vol. 7, pp. 85 727–85 745, 2019.

[75] IBM, "Ibm blockchain platform technical overview." [Online]. Available: https://www.ibm.com/downloads/cas/Q9DGBLV7

[76] A. Al Badawi *et al.*, "Openfhe: Open-source fully homomorphic encryption library," in *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2022, pp. 53–63.

[77] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[78] Go-Ethereum, "Official go implementation of the ethereum protocol," Accessed: August. 13, 2024. [Online]. Available: https://github.com/ethereum/go-ethereum

[79] Prysmatic-Lab, "Prysm: An ethereum consensus implementation written in go," Accessed: August. 13, 2024. [Online]. Available: https://github.com/prysmaticlabs/prysm