國立臺北科技大學
National Taipei University of Technology

資訊安全碩士學位學程

碩士學位論文

Master of Science in Information Security

Master Thesis

基於 K-min 控制機制之非同步聯邦學習入侵偵測方法

K-min Controlled Asynchronous Federated Learning for Intrusion Detection

研究生：雍皓崴

Researcher：Hao-Wei Yong

指導教授：陳香君 博士

Advisor：Shiang-Jiun Chen, Ph.D.

June 2026

國立臺北科技大學
研究所碩士學位論文口試委員會審定書

本校 資訊安全碩士學位學程 研究所 雍皓崴 君

所提論文，經本委員會審定通過，合於碩士資格，特此證明。

學位考試委員會

委　　員：陳香君

馬奕薇

孫文輝

指導教授：陳香君

所　　長：

中 華 民 國　一百一十五　年　6　月　12　日

Abstract

Keywords: Asynchronous Federated Learning, Non-IID Data, Intrusion Detection System, IoT Security, Aggregation Control.

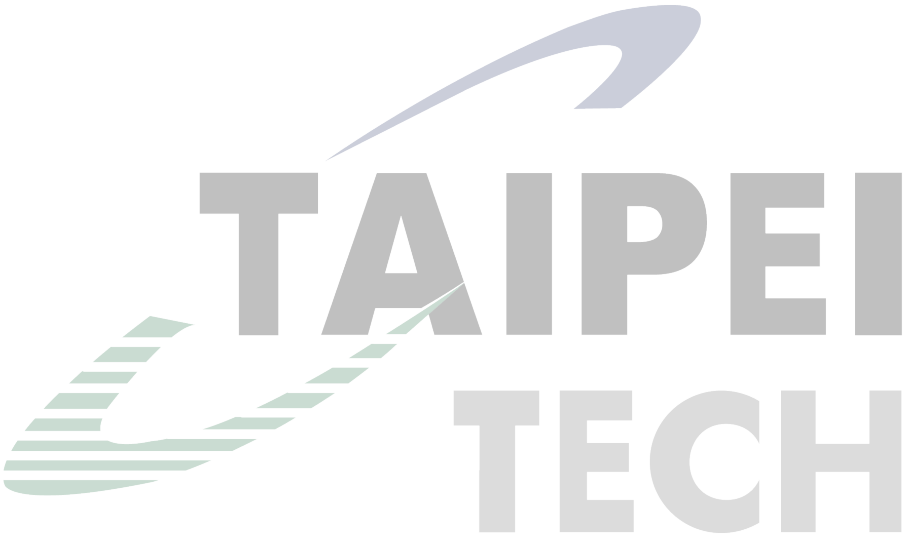
With the rapid growth of Internet of Things (IoT) devices, a large number of heterogeneous nodes are deployed at the network edge, posing significant challenges for intrusion detection, including data heterogeneity (Non-IID), limited computational resources, and high communication overhead. Traditional centralized intrusion detection systems require aggregating data at a central server, which introduces privacy risks and communication bottlenecks. While synchronous federated learning suffers from straggler effects that reduce training efficiency, asynchronous federated learning alleviates waiting time but may lead to unstable model performance under highly Non-IID conditions.

To address these challenges, this study proposes a K-min Aggregation-Controlled Federated Learning (KAC-FL) framework, which integrates asynchronous federated learning with a K-min aggregation control mechanism. By adjusting the minimum number of participating gateways required for aggregation, the proposed approach balances aggregation frequency and model performance. In addition, a participation priority mechanism is introduced to mitigate participation imbalance among gateways. Furthermore, a data-size-based weighted aggregation strategy is adopted to reduce the impact of Non-IID data distributions.

Experiments are conducted on the CICIDS2017 dataset with multiple gateways simulating heterogeneous IoT edge environments. The results show that the proposed KAC-FL framework achieves high detection performance while improving training efficiency under different aggregation settings. Compared with the fully asynchronous setting ($K_{\min} = 1$), the KAC-FL framework significantly improves F1-score and Recall while maintaining high AUC performance. Compared with the near-synchronous setting ($K_{\min} = 10$), the proposed approach substantially reduces total training time while preserving competitive detection performance.

Overall, the proposed KAC-FL framework demonstrates an effective trade-off between

detection performance and training efficiency for intrusion detection in heterogeneous Non-IID IoT edge environments.



Acknowledgements

首先，衷心感謝我的指導教授陳香君教授。在碩士求學期間，老師不僅在研究方向與論文撰寫上給予悉心指導，也在遇到困難與瓶頸時提供寶貴的建議與鼓勵，使我能夠逐步完成研究工作並順利完成本論文。老師嚴謹的治學態度與對研究品質的要求，讓我在學術研究與問題分析能力上獲益良多，謹此致上最誠摯的敬意與感謝。

此外，感謝口試委員馬奕葳教授與謝文彬教授撥冗審閱本論文，並於口試過程中提供寶貴意見與建議，使本論文內容得以更加完整與完善，在此致上由衷的謝意。

感謝實驗室的同學凱俊、子杰、佳蓉、榕佑、雋洋、少雷、承展、兆晞、鈺翔及昌澔，在研究與求學期間給予許多協助與陪伴。無論是在課業討論、研究交流、實驗測試，或是日常生活中的互相鼓勵，都讓我的研究生活增添許多美好的回憶。也感謝學長正承、祥語、劭睿、承諺及子榆，在研究方向、論文撰寫與經驗分享方面提供許多寶貴的建議，讓我在面對研究問題時能夠少走許多彎路。

最後，我要將最深的感謝獻給我的家人。感謝爸爸、媽媽以及哥哥一路以來無條件的支持與鼓勵，無論是在求學、生活或人生道路上的每一個重要階段，都始終給予我最大的包容與後盾。正因爲有你們的陪伴與支持，我才能夠專心完成學業並順利完成碩士論文。

碩士生涯即將告一段落，這段期間所獲得的知識、經驗與情誼，將成爲我人生中珍貴的資產。謹以此篇論文，獻給所有曾經幫助、支持與陪伴我的師長、同學、朋友及家人，並向各位致上最誠摯的感謝。

雍皓崴謹誌於

臺北科技大學資訊安全碩士學位學程班

中華民國 115 年 6 月 12 日

TAIPEI TECH

Table of Contents

Abstract i

Acknowledgements iii

Table of Contents iv

List of Figures vi

List of Tables vii

Chapter 1 Introduction 1

Chapter 2 Related Work 4

2.1 Intrusion Detection in IoT Environment 4

2.2 Machine Learning-Based Intrusion Detection System 6

2.3 Federated Learning for Distributed Intrusion Detection 7

2.3.1 Federated Learning 7

2.3.2 Asynchronous Federated Learning 8

2.4 Aggregation Strategies 10

2.5 Data Heterogeneity in Federated Learning 11

Chapter 3 KAC-FL Methods 13

3.1 KAC-FL Architecture 13

3.1.1 Gateway Layer 14

3.1.2 Controller Layer 15

3.1.3 Cloud Layer 16

3.2 KAC-FL Operation Process 17

3.2.1 Gateway Layer Operation Process 19

3.2.2 Controller Layer Operation Process 22

3.2.3 Cloud Layer Operation Process 24

Chapter 4 Implementation 28

4.1 Experimental Environment 28

4.1.1 Hardware Deployment Configuration 28

4.1.2 Software Environment Configuration 29

TAIPEI TECH

4.2 Implementation Details . . . 30
4.3 Dataset Preparation and Non-IID Distribution . . . 31
4.3.1 Dataset Split . . . 32
4.3.2 Gateway Non-IID Allocation . . . 32
4.4 Experimental Configuration . . . 33
4.5 Implementation Steps . . . 35
4.5.1 Environment Initialization . . . 35
4.5.2 Distributed Training Execution . . . 36
Chapter 5 Results and Analysis . . . 42
5.1 Evaluation Metrics . . . 42
5.2 Detection Performance Analysis . . . 45
5.2.1 F1-score Analysis . . . 45
5.2.2 Recall Analysis . . . 46
5.2.3 AUC Analysis . . . 47
5.3 Training Efficiency Analysis . . . 48
5.3.1 Total Runtime Analysis . . . 48
5.3.2 Aggregation Time Analysis . . . 50
5.4 Comparative Analysis and Discussion . . . 51
5.5 Summary . . . 53
Chapter 6 Conclusion and Future Work . . . 55
6.1 Conclusion . . . 55
6.2 Future Work . . . 56
References . . . 58

TAIPEI TECH

List of Figures

2.1 System design of FLIP4 [2] 5

2.2 Workflow of Federated Learning [3] 8

2.3 Architecture of the WKAFL framework [17] 11

3.1 Architecture of the K-min Aggregation-Controlled Federated Learning(KAC-FL) 13

3.2 Gateway Layer 14

3.3 Controller Layer 15

3.4 Cloud Layer 16

3.5 KAC-FL Workflow 18

3.6 Gateway initialization workflow. 19

3.7 Gateway iterative training workflow. 20

3.8 Controller workflow. 22

3.9 Cloud workflow. 24

5.1 Impact of K_{min} on final average test F1-score. 45

5.2 Impact of K_{min} on final average test recall. 46

5.3 Impact of K_{min} on final average test AUC. 47

5.4 Impact of K_{min} on total runtime. 49

5.5 Impact of K_{min} on average aggregation time. 50

TAIPEI TECH

List of Tables

4.1 Hardware deployment configuration of the experimental environment. 28

4.2 Software environment configuration of the experimental environment. 29

4.3 Gateway VM software dependencies. 30

4.4 Controller VM software dependencies. 31

4.5 Cloud VM software dependencies. 31

4.6 Dataset split configuration. 32

4.7 Non-IID data allocation across gateway nodes. 33

4.8 Experimental parameter configuration. 34

5.1 Representative performance comparison under different aggregation threshold settings 51

TAIPEI TECH

Chapter 1 Introduction

The rapid growth of Internet of Things (IoT) technologies has enabled the deployment of massive numbers of interconnected devices in smart manufacturing, smart healthcare, intelligent transportation, and smart city applications. These IoT devices continuously generate large volumes of heterogeneous network traffic, making intrusion detection an essential mechanism for maintaining system security. However, most IoT devices are constrained by limited computational capability, storage capacity, and energy resources, making them unsuitable for executing computationally intensive machine learning models locally. Consequently, modern IoT systems commonly adopt an edge computing architecture [1], in which multiple IoT devices are connected to nearby edge gateways. Instead of transmitting all raw network traffic directly to the cloud, the gateway performs local data preprocessing and local model training, while computationally intensive tasks such as global model aggregation, model management, and coordination among distributed gateways are handled by the cloud server [2]. This edge-cloud collaborative architecture significantly reduces communication overhead and network latency while preserving data privacy and improving scalability for large-scale IoT deployments.

Federated Learning (FL) has therefore emerged as a promising distributed learning paradigm for IoT intrusion detection [3]. Instead of collecting raw traffic data at a centralized server, each edge gateway trains a local intrusion detection model using locally collected traffic and only shares model parameters with the cloud for global aggregation. By keeping raw data locally, FL effectively addresses privacy concerns while reducing communication costs and enabling collaborative learning across geographically distributed gateways. Nevertheless, practical IoT environments introduce another critical challenge. Since each gateway typically serves different geographical regions, device types, user behaviors, and network services, the traffic distributions among gateways are naturally heterogeneous rather than independently and identically distributed (Non-IID). Such heterogeneous data distributions often degrade model convergence and reduce the generalization capability of conventional federated learning algorithms [4].

To alleviate the communication delay introduced by synchronous federated learning, Asynchronous Federated Learning (AFL) allows the cloud server to aggregate newly received local

models without waiting for every gateway to complete training [5]. Although AFL improves training efficiency and mitigates the straggler problem, excessively frequent aggregation under highly Non-IID environments may result in unstable convergence and performance degradation because global models are updated using insufficient or highly biased local information. Therefore, balancing aggregation efficiency and model stability has become an important research issue for asynchronous federated learning in heterogeneous IoT edge environments.

To address these challenges, this study proposes a K-min Aggregation-Controlled Federated Learning (KAC-FL) framework for intrusion detection in heterogeneous IoT edge environments. The proposed framework introduces a configurable K-min aggregation control mechanism, which performs global aggregation only after receiving a minimum number of local model updates, thereby improving aggregation stability under Non-IID data distributions. Furthermore, a participation priority mechanism is designed to reduce gateway participation imbalance, while a data-size-based weighted aggregation strategy is adopted to alleviate the influence of heterogeneous local datasets. Experimental results on the CICIDS2017 dataset demonstrate that the proposed framework effectively balances intrusion detection performance and training efficiency under different aggregation settings.

TAIPEI TECH

Contributions

The main contributions of this work are summarized as follows:

- This study proposes a K-min Aggregation-Controlled Federated Learning (KAC-FL) framework for distributed intrusion detection in heterogeneous IoT environments. By integrating asynchronous federated learning with configurable K-min controlled aggregation, the proposed framework enables flexible synchronization control to balance intrusion detection effectiveness and execution efficiency. Experimental results show that, compared with the fully asynchronous setting ($K_{\text{min}} = 1$), an intermediate configuration ($K_{\text{min}} = 3$) improves F1-score by approximately 32% and Recall by approximately 66.7%. Compared with the near-synchronous setting ($K_{\text{min}} = 10$), a balanced configuration ($K_{\text{min}} = 6$) reduces total runtime by approximately 91.4%, with only approximately 10% F1-score

reduction and 13.3% Recall reduction while maintaining competitive AUC performance.

- This study further introduces a participation priority mechanism to mitigate gateway participation imbalance in asynchronous federated learning. By prioritizing previously excluded gateways during the late stage of training, the proposed mechanism improves gateway inclusion opportunities and supports more balanced participation under heterogeneous distributed IoT environments.

The remainder of this thesis is organized as follows. Chapter 2 reviews the related work on IoT intrusion detection, federated learning, asynchronous federated learning, and aggregation strategies relevant to this study. Chapter 3 presents the proposed KAC-FL framework, including the overall system architecture, operational workflow, and the design of the K-min controlled aggregation and participation priority mechanisms. Chapter 4 describes the implementation details of the proposed framework, including the experimental environment, software configuration, dataset preparation, and deployment workflow. Chapter 5 presents the experimental results and performance analysis, evaluating the proposed framework in terms of intrusion detection effectiveness and execution efficiency under different aggregation settings. Finally, Chapter 6 presents the conclusions of this study and discusses potential directions for future research.

TAIPEI TECH

Chapter 2 Related Work

This chapter presents a review of the technical background and related work associated with this study. It begins by examining intrusion detection architectures in IoT environments, followed by machine learning-based intrusion detection approaches. Subsequently, federated learning [6] paradigms, including synchronous and asynchronous [5] training mechanisms, are reviewed. In addition, commonly adopted aggregation strategies in federated learning are discussed, along with the challenges introduced by heterogeneous data distributions. These studies provide the technical foundation for understanding the design considerations of the proposed research.

2.1 Intrusion Detection in IoT Environment

Ge et al. [7] proposed a deep learning-driven intrusion detection framework for IoT environments to address the increasing security threats faced by resource-constrained IoT devices. Their study emphasized that traditional security mechanisms are often insufficient for heterogeneous IoT deployments due to limited computational resources and dynamic traffic behaviors. By designing an intelligent intrusion detection architecture capable of analyzing network traffic patterns, the proposed approach demonstrated the feasibility of deploying advanced intrusion detection mechanisms in practical IoT environments.

Otoum et al. [8] proposed a collaborative intrusion detection framework for distributed IoT environments, where multiple monitoring nodes cooperate to improve attack detection coverage. Instead of relying on a centralized detection mechanism, their approach distributes monitoring responsibilities across multiple network components, enabling scalable intrusion detection in large-scale IoT deployments. Experimental results showed that collaborative detection improves detection effectiveness and reduces the limitations of single-node monitoring architectures.

Zang et al. [2] proposed Federated Learning-Based In-Network Packet Processing Platform(FLIP4) for IoT edge environments. The framework combines federated learning with programmable data planes to support distributed attack detection while preserving data locality.

Instead of transmitting raw traffic data to a centralized server, FLIP4 keeps traffic data locally at IoT edge gateways and shares local model information for global aggregation, thereby reducing communication overhead and privacy risks.

As illustrated in Fig. 2.1, the FLIP4 architecture consists of IoT end devices, programmable edge gateways, and a cloud aggregation server. Each gateway trains a local tree-based intrusion detection model using its own traffic data, maps the trained model into the programmable data plane for in-network inference, and sends model parameters to the cloud server with differential privacy protection. The cloud then aggregates the received local models and redistributes the updated global model back to the gateways for runtime model updates.

Experimental results showed that FLIP4 achieved accurate attack detection on the CI-CIDS2017 dataset while maintaining source data privacy and enabling lightweight deployment on IoT edge. However, FLIP4 primarily focuses on federated tree-based in-network inference and model deployment, while synchronization control and aggregation triggering under heterogeneous asynchronous gateway environments are not the main focus of the framework.

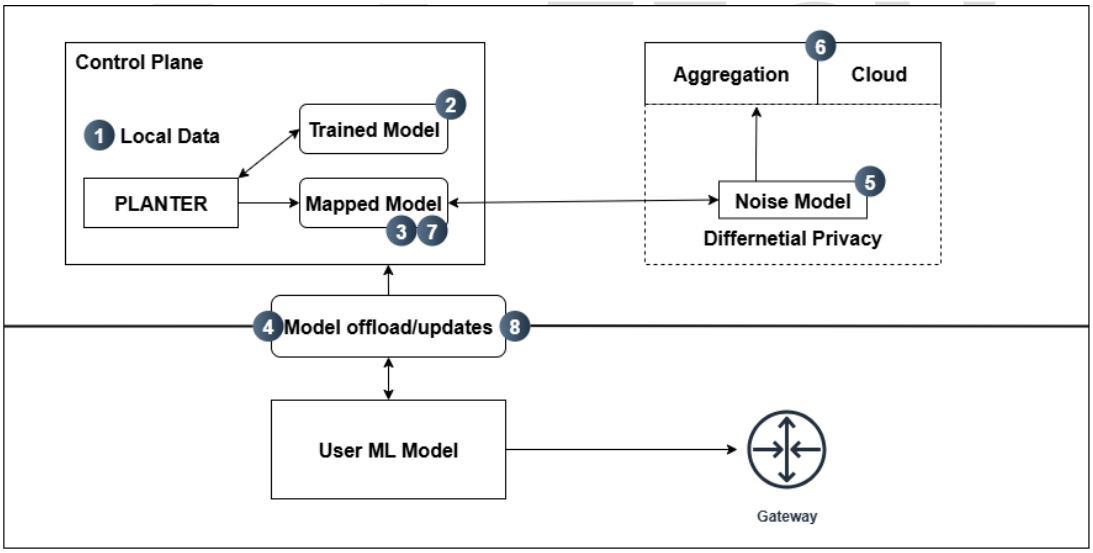


Figure 2.1 System design of FLIP4 [2]

2.2 Machine Learning-Based Intrusion Detection System

Extreme Gradient Boosting (XGBoost) is a scalable machine learning algorithm based on the gradient tree boosting framework. In [9], Chen and Guestrin developed XGBoost to address regression and classification problems with high computational efficiency and prediction accuracy. The algorithm constructs an ensemble of decision trees in a sequential manner, where each subsequent tree is trained to correct the errors made by the previous trees through gradient-based optimization. XGBoost incorporates several advanced techniques including a regularized learning objective that combines a convex loss function with regularization terms to prevent overfitting, an efficient split-finding algorithm for handling large-scale datasets, and built-in mechanisms for managing missing values and sparse data. These characteristics make XGBoost particularly well-suited for network intrusion detection tasks involving structured tabular network traffic data.

In [10], a study on the CICIDS2017 dataset showed that XGBoost achieved the highest detection accuracy of 99% among decision tree family algorithms, outperforming Random Forest with 98%, Gradient Boosting Trees with 88%, and standard Decision Tree with 89%.

In [11], an ensemble learning approach combining XGBoost with autoencoders achieved an impressive detection accuracy of 99.92% on the CICIDS2017 dataset, demonstrating strong performance across multiple attack categories including DDoS, PortScan, and botnet activities.

These results highlight XGBoost's suitability for federated learning-based intrusion detection frameworks, where computational efficiency, compact model representation, and high detection accuracy are essential requirements for deployment on distributed edge devices.

TAIPEI TECH

2.3 Federated Learning for Distributed Intrusion Detection

2.3.1 Federated Learning

Federated learning emerged as a distributed machine learning approach that enables collaborative model training across multiple decentralized devices without requiring raw data centralization [6]. This paradigm fundamentally differs from conventional centralized learning architectures, where data aggregation at a single location often leads to privacy vulnerabilities and substantial transmission costs. The key innovation of FL lies in its ability to maintain data locality while still achieving collective intelligence through parameter sharing rather than data sharing.

McMahan et al. [3] proposed Federated Averaging (FedAvg), one of the most widely adopted federated learning algorithms. FedAvg enables multiple distributed clients to collaboratively train a shared global model without directly exchanging raw training data. Instead, each client performs local model training using its own dataset and periodically uploads model updates to a central server for aggregation.

As illustrated in Fig. 2.2, the federated learning process begins with the server distributing the current global model to participating clients. Each client independently performs local training using its local dataset and generates a model update. The server then collects updates from all participating clients and computes a weighted average based on the local data size of each client to produce a new global model. The updated global model is subsequently redistributed to clients for the next training round.

FedAvg significantly reduces privacy risks by keeping training data locally and only sharing model parameters during the learning process. However, the algorithm follows a fully synchronous training paradigm, requiring the server to wait for all selected clients before performing aggregation. As the number of participants increases or client capabilities become heterogeneous, synchronization delays and straggler effects may significantly increase the overall train-

TAIPEI TECH

ing time.

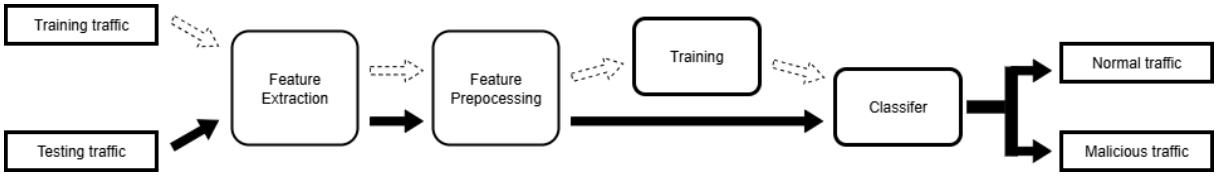


Figure 2.2 Workflow of Federated Learning [3]

Li et al. [12] proposed FedProx, an extension of synchronous federated learning designed to improve robustness under heterogeneous network and device conditions. Similar to FedAvg, FedProx follows a synchronous training paradigm in which selected clients perform local training before sending updates to a central server for aggregation. To address client heterogeneity, the method introduces a proximal term in the local optimization objective, improving convergence stability under varying computational capabilities and non-identically distributed data. Despite these improvements, the synchronous aggregation mechanism still requires waiting for slower clients, resulting in increased communication latency and reduced training efficiency in large-scale distributed environments.

Synchronous federated learning provides stable and consistent global model updates, making it a widely adopted baseline in federated learning research. However, its dependence on waiting for all participating clients introduces communication bottlenecks and reduced scalability in heterogeneous environments, particularly in IoT scenarios where device capabilities and network conditions vary significantly.

2.3.2 Asynchronous Federated Learning

Chen et al. [13] proposed FedSA, a staleness-aware asynchronous federated learning algorithm designed for Non-IID data environments. The method addresses two major challenges in AFL: unstable model performance caused by Non-IID data distributions and staleness issues caused by unreliable or slow client environments. Instead of treating all client updates equally, FedSA considers the staleness of received updates during aggregation, reducing the negative impact of outdated local models on the global model. Experimental results showed that FedSA improves training stability and convergence performance compared with conventional asyn-

chronous federated learning methods.

Nguyen et al. [14] proposed FedBuff, a buffered asynchronous federated learning framework that combines the scalability of asynchronous FL with the aggregation behavior of synchronous FL. Instead of updating the global model immediately after receiving a single client update, FedBuff stores client updates in a server-side buffer and performs aggregation once the buffer condition is satisfied. This design improves scalability and reduces the drawbacks of fully asynchronous aggregation, while also supporting privacy-preserving mechanisms such as secure aggregation and differential privacy. Experimental results showed that FedBuff achieved higher efficiency than synchronous FL and conventional asynchronous FL.

Zhang et al. [15] proposed TimelyFL, a heterogeneity-aware asynchronous federated learning framework with adaptive partial training. The study pointed out that in cross-device FL environments, synchronous FL suffers from straggler effects, while existing asynchronous FL methods may suffer from participation imbalance, where fast clients contribute more frequently and slow clients participate less often. To address this issue, TimelyFL adjusts local training workloads according to the real-time resource capabilities of clients, allowing more clients to participate in global updates with reduced staleness. Experimental results showed that TimelyFL improves participation rate, convergence efficiency, and test accuracy compared with FedBuff.

Liu et al. [16] proposed FedASMU, an asynchronous staleness-aware model update framework designed to address both statistical heterogeneity and system heterogeneity in federated learning. The method introduces a dynamic model aggregation strategy between updated local models and the global model on the server, aiming to improve accuracy and training efficiency under heterogeneous environments. By considering the staleness of local models during aggregation, FedASMU reduces the negative influence of outdated updates and accelerates convergence. Experimental results showed that FedASMU achieved faster convergence than several asynchronous FL baselines, including FedAsync, FedBuff, and FedSA.

TAIPEI TECH

2.4 Aggregation Strategies

Zhou et al. [17] proposed Weighted K-Asynchronous Federated Learning (WKAFL) to improve convergence stability and training efficiency in asynchronous federated learning under Non-IID environments. Unlike conventional asynchronous federated learning, which updates the global model immediately after receiving a single client update, WKAFL introduces a K-asynchronous aggregation mechanism that waits for K client updates before performing global aggregation. In addition, WKAFL incorporates weighted aggregation, gradient selection, and adaptive learning-rate adjustment to mitigate the impact of stale gradients and heterogeneous data distributions.

As illustrated in Fig. 2.3, the framework consists of five main components, including gradient computation at clients, Non-IID mitigation, globally unbiased gradient estimation, consistent gradient selection and aggregation, and learning-rate adaptation at the server side. Experimental results demonstrated that WKAFL achieved better convergence stability, prediction accuracy, and training efficiency compared with conventional asynchronous federated learning approaches. However, WKAFL mainly focuses on stale-gradient mitigation and model convergence, while the participation fairness of clients during asynchronous aggregation is not explicitly considered.

TAIPEI TECH

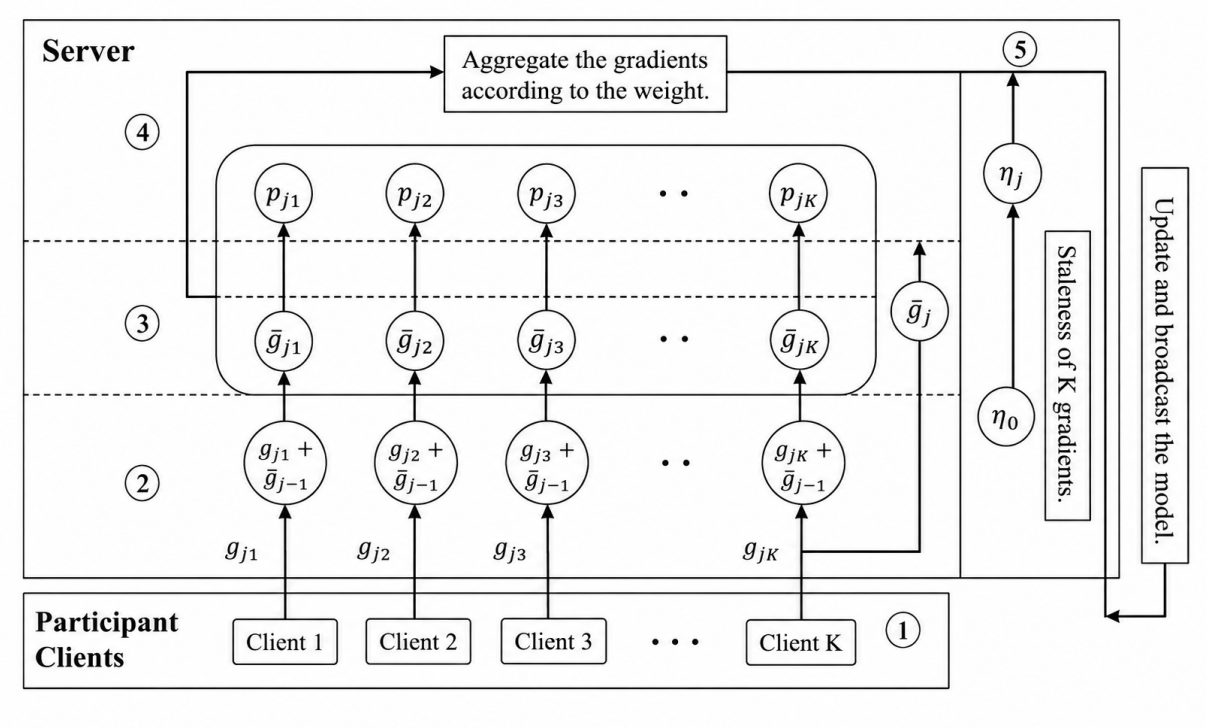


Figure 2.3 Architecture of the WKAFL framework [17]

In [18], Wen et al. proposed a Dynamic Weighted K-Asynchronous Federated Learning (DWKAFL) approach for smart grid intrusion detection systems. The method employs K-asynchronous aggregation where the central server performs model updates after receiving updates from a variable number of clients rather than waiting for all clients to complete training. To determine aggregation weights, the approach considers both gradient quality and staleness metrics, allowing the system to prioritize contributions from clients with higher-quality updates while mitigating the impact of delayed submissions. An Adaptive Gradient Storage Bucket (AGSB) scheme is utilized to store gradients based on their arrival times and optimize the timing of aggregation tasks. Experimental evaluation on the CICIDS2017 dataset demonstrated an accuracy of 91.3%, showing the feasibility of combining asynchronous training with quality-aware aggregation strategies in network intrusion detection scenarios.

2.5 Data Heterogeneity in Federated Learning

In federated learning systems, data across participating clients is typically non-independent and identically distributed (Non-IID) [19], presenting significant challenges for model training

and convergence. In [20], Jimenez et al. provided a comprehensive taxonomy for Non-IID data in federated learning environments, categorizing data heterogeneity into several distinct types including label distribution skew, feature distribution skew, and quantity skew. Label distribution skew occurs when different clients possess data samples with varying class distributions, resulting in local models that may be biased toward specific categories. Feature distribution skew arises when clients observe different subsets of the overall feature space, common in scenarios where devices operate in distinct environmental conditions or user contexts. Quantity skew refers to imbalanced data volumes across clients, where some participants may have significantly more training samples than others. These forms of data heterogeneity can severely impact federated learning performance, leading to slower convergence rates, degraded model accuracy, and reduced generalization capabilities compared to scenarios with IID data distributions. The study emphasized that Non-IID data presents additional challenges for model parameter aggregation, as local model updates may diverge significantly from the global optimization direction. Furthermore, the presence of Non-IID data necessitates specialized aggregation strategies and model training techniques to ensure that the global model effectively captures patterns from all clients while mitigating the negative effects of statistical heterogeneity.

Based on the reviewed literature, intrusion detection in IoT environments has gradually evolved from centralized architectures toward distributed and collaborative approaches to better accommodate resource-constrained devices and heterogeneous deployment conditions. Machine learning techniques have significantly improved intrusion detection capability, while federated learning further enables collaborative model training without requiring direct data sharing. However, existing federated learning approaches still face several challenges in practical IoT environments, including communication inefficiency in synchronous training, instability caused by stale updates in asynchronous training, and performance degradation under heterogeneous data distributions. These challenges highlight the need for a more efficient and stable federated learning framework tailored for distributed IoT intrusion detection scenarios. Therefore, the next chapter presents the proposed system architecture and methodology of this research.

Chapter 3 KAC-FL Methods

This chapter presents a comprehensive description of the K-min Aggregation-Controlled Federated Learning (KAC-FL) architecture, detailing its layered design as well as the corresponding modules and their functionalities. For clarity, each layer is discussed in a dedicated subsection. Furthermore, the chapter concludes with a summary of the overall operational flow of KAC-FL.

3.1 KAC-FL Architecture

The proposed KAC-FL (K-min Aggregation-Controlled Federated Learning) framework is designed for distributed intrusion detection in heterogeneous IoT edge environments. As illustrated in Fig 3.1, the system adopts a three-layer architecture consisting of the Gateway Layer, Controller Layer, and Cloud Layer. Each layer is responsible for a specific set of functionalities, enabling efficient collaboration among distributed devices while maintaining scalability, security, and robustness.

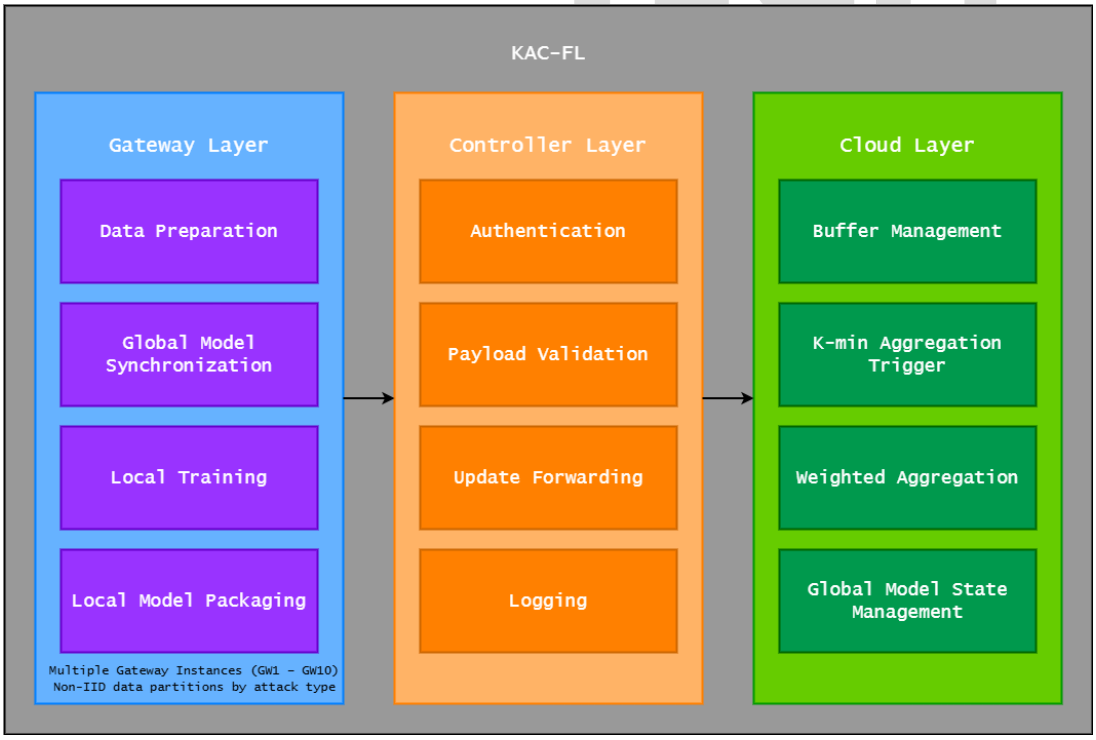


Figure 3.1 Architecture of the K-min Aggregation-Controlled Federated Learning(KAC-FL)

3.1.1 Gateway Layer

The Gateway Layer represents distributed edge nodes deployed in IoT environments, as shown in Fig 3.2. Each gateway is responsible for local data processing, model training, and communication with the upper layers. In this study, multiple gateway instances (GW1—GW10) are used, where local datasets are partitioned in a Non-IID manner based on attack types to simulate realistic IoT scenarios.

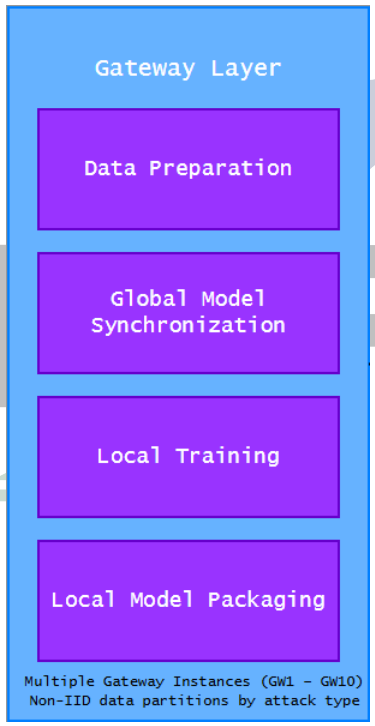


Figure 3.2 Gateway Layer

The Gateway Layer consists of the following modules:

- Data Preparation Module: Responsible for loading raw network traffic data, performing preprocessing, and generating training samples.
- Global Model Synchronization Module: Retrieves the initial and latest global models from the cloud and ensures consistency between local and global model versions.
- Local Training Module: Performs model training using XGBoost [9] based on local data and the received global model.

- Local Model Packaging Module: Packages the trained local model along with auxiliary information (e.g., number of samples, gateway identifier, and model version) into a structured format for transmission.

After local training, each gateway sends its local model package to the Controller Layer and subsequently retrieves the latest global model to continue the next training round.

3.1.2 Controller Layer

The Controller Layer acts as an intermediate management layer between gateways and the cloud. As shown in Fig 3.3, its primary role is to ensure secure and valid communication without participating in model training or aggregation.

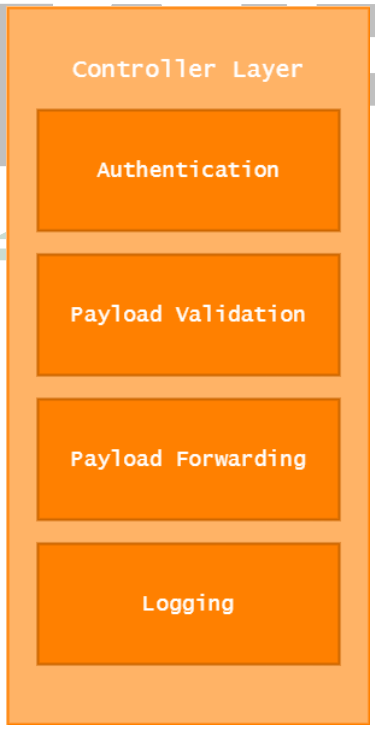


Figure 3.3 Controller Layer

The Controller Layer includes the following modules:

- Authentication Module: Verifies the identity of each gateway to ensure that only authorized devices can participate in the system.
- Payload Validation Module: Validates the structure and integrity of the received local model package, ensuring correctness before forwarding.

- Logging Module: Records system events, including successful transmissions and rejected requests, for traceability and monitoring.
- Payload Forwarding Module: Forwards validated local model packages to the Cloud Layer for further processing.

  If a local model package fails authentication or validation, it is rejected and logged accordingly.

3.1.3 Cloud Layer

The Cloud Layer is the core component of the KAC-FL framework, responsible for model management, as shown in Fig 3.4, aggregation control, and global coordination. It maintains the global model state and orchestrates the aggregation process based on the K-min mechanism.

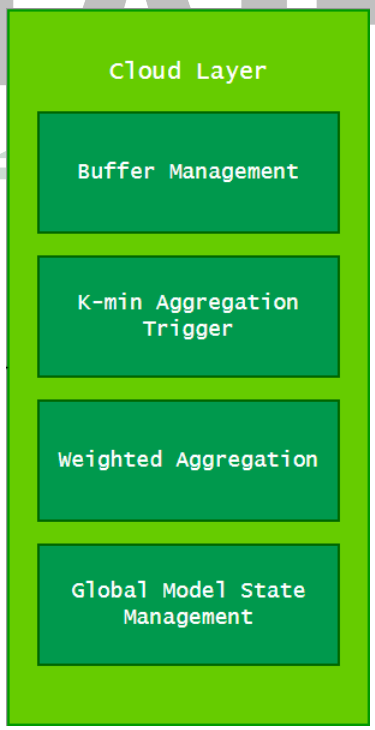


Figure 3.4 Cloud Layer

The Cloud Layer consists of the following modules:

- Buffer Management Module: Stores incoming local model packages and ensures that only the latest model from each gateway is retained. It also maintains the buffer capacity and removes outdated or already-used entries.

- K-min Aggregation Trigger Module: Determines whether the number of available local models reaches the predefined threshold K-min Aggregation is triggered only when this condition is satisfied.

- Weighted Aggregation Module: Performs aggregation of selected local models using a weighted scheme (e.g., based on the number of samples), addressing data imbalance under Non-IID settings.

- Global Model State Management Module: Updates and stores the global model after aggregation, maintains the global round counter, and provides the latest model to gateways.

Additionally, a late-stage control mechanism is incorporated to ensure participation fairness. When the system approaches the final training rounds, the cloud ensures that gateways with no prior participation are included in the aggregation process, thereby reducing the possibility that certain gateways remain excluded from aggregation.

The above describes the architecture and individual modules of the K-min Aggregation-Controlled Federated Learning (KAC-FL). In the following Section 3.2, the operational workflow of KAC-FL will be introduced in detail.

3.2 KAC-FL Operation Process

The operational workflow of the proposed KAC-FL framework is illustrated in Fig. 3.5. The framework adopts a three-layer architecture consisting of the Gateway Layer, Controller Layer, and Cloud Layer, where each layer is responsible for distinct functional operations during the federated training process. The Gateway Layer handles local data processing and model training, the Controller Layer performs request authentication, payload validation, and logging, while the Cloud Layer coordinates aggregation and global model management. To provide a clearer understanding of the implementation details, the overall workflow is further decomposed into four layer-specific operational processes, which are described in the following subsections.

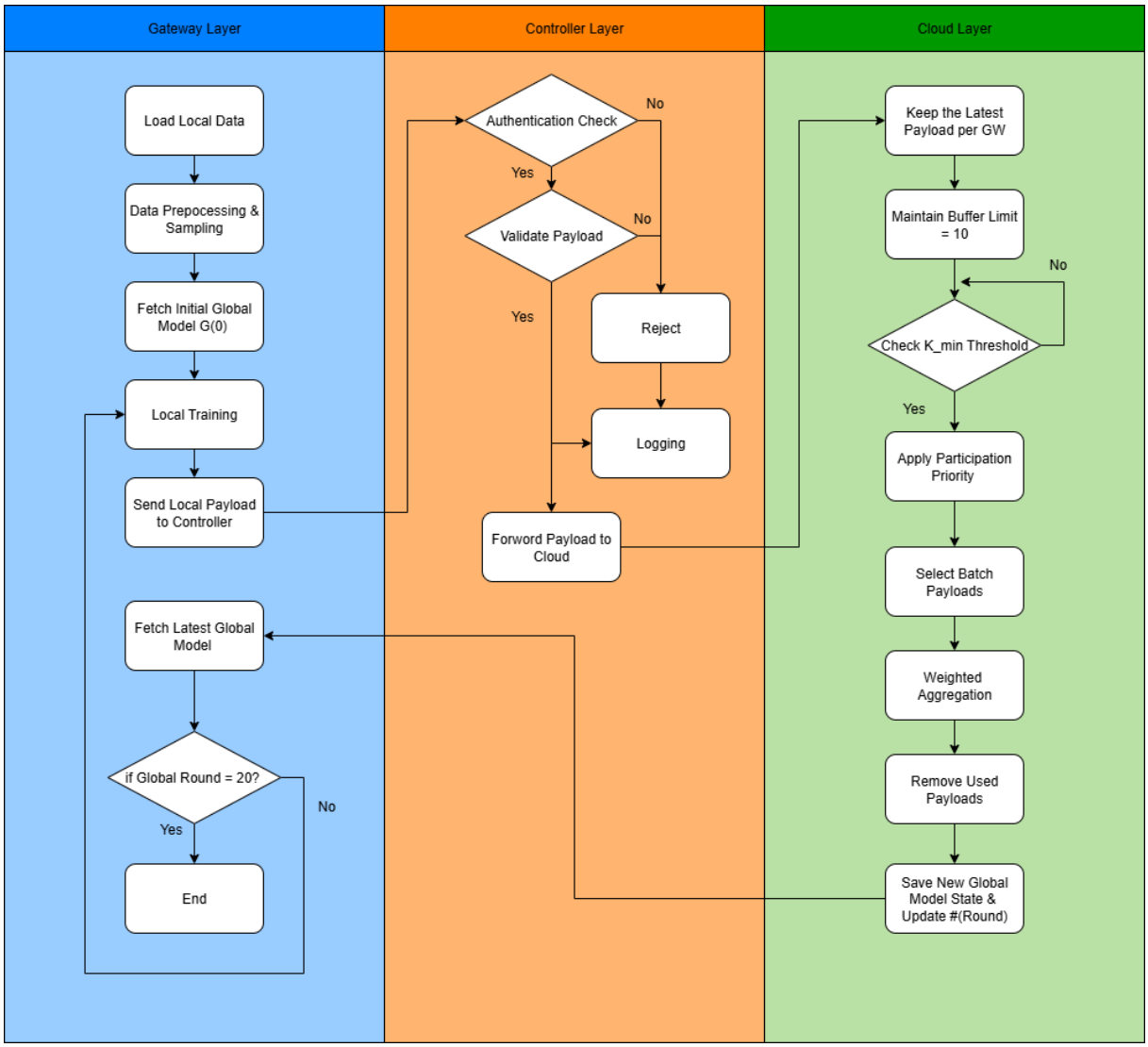


Figure 3.5 KAC-FL Workflow

3.2.1 Gateway Layer Operation Process

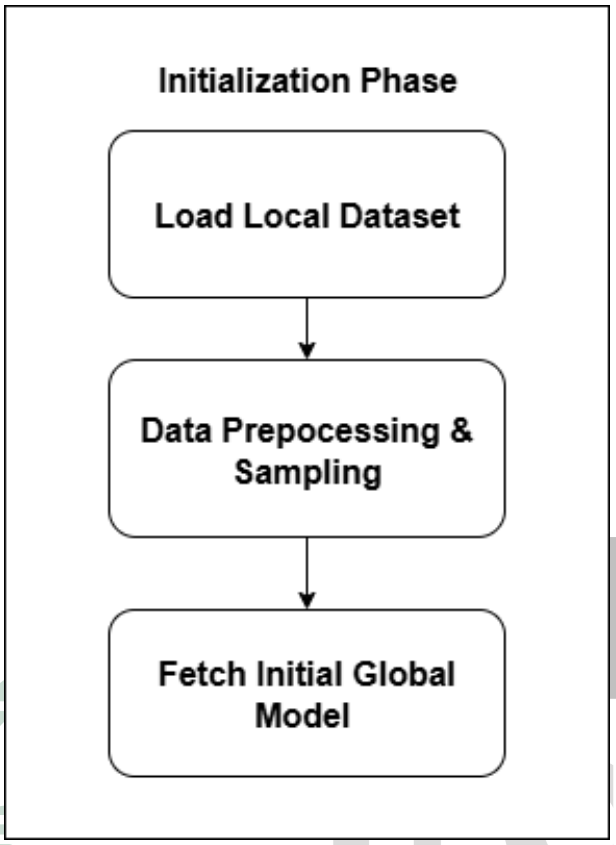


Figure 3.6 Gateway initialization workflow.

As shown in Fig. 3.6, the gateway operation begins with the initialization phase. Each gateway first loads its local dataset. In this study, the CICIDS2017 dataset [21] is used as the experimental dataset because it contains both benign and malicious network traffic records, making it suitable for intrusion detection tasks. To simulate heterogeneous IoT edge environments, the dataset is distributed across multiple gateways in a Non-IID manner. Each gateway therefore owns a local dataset with different traffic characteristics, attack types, and class distributions.

After loading the local dataset, the gateway performs data preprocessing and sampling. The preprocessing process includes removing invalid or missing values, converting non-numeric features into usable numerical formats, handling infinite values, and transforming the original multi-class attack labels into binary labels. Specifically, benign traffic is labeled as 0, while malicious traffic is labeled as 1. This binary classification setting is used to train the local intrusion detection model. In addition, sampling is applied to construct different local data

distributions among gateways. This step is used to reflect the practical condition where different gateways in IoT environments observe different traffic patterns due to differences in device type, deployment location, and network behavior.

After the local dataset is processed, the gateway obtains the initial global model $G(0)$ from the cloud server. This model serves as the starting point for local training.

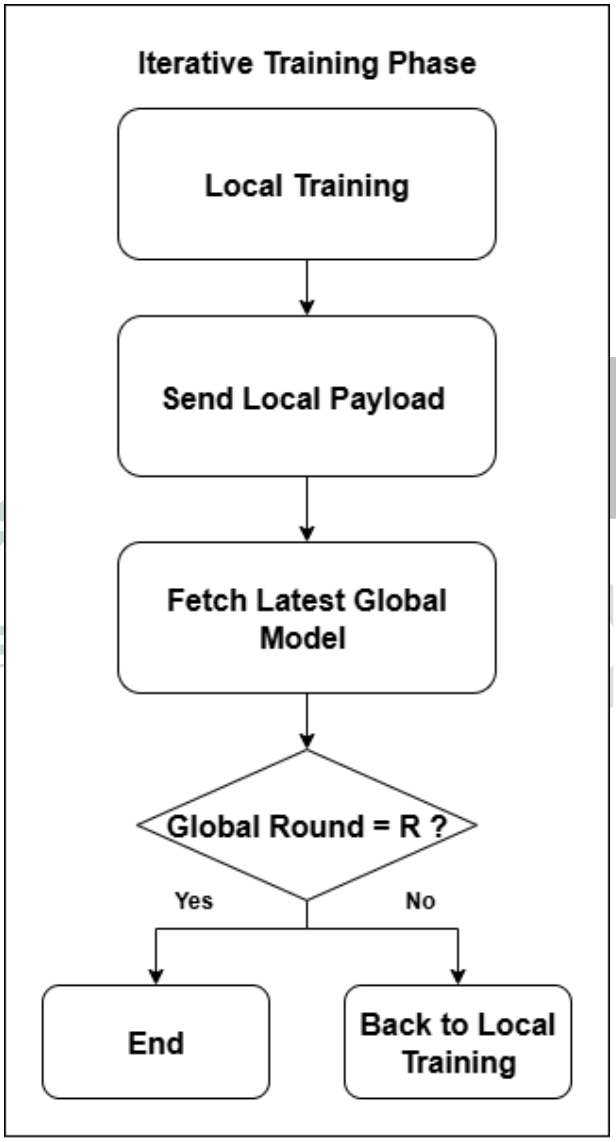


Figure 3.7 Gateway iterative training workflow.

Once the initialization phase is completed, the gateway enters the iterative training phase, as shown in Fig. 3.7. In the iterative training phase, the gateway performs local model training using XGBoost [9]. XGBoost is adopted as the local learning model because it is effective for structured tabular data and provides efficient training performance for network traffic classifica-

tion. In each global round, the gateway trains an XGBoost-based local intrusion detection model using its processed local dataset. Since each gateway has a different local data distribution, the trained local model reflects the traffic characteristics observed by that gateway.

After local training is completed, the gateway generates a local payload and sends it to the Controller Layer. The payload does not include raw traffic data. Instead, it contains the trained local model information and required metadata, including the gateway identifier, model version, source global round, local training sample size, training time, and local evaluation metrics. The local training sample size n_i is particularly important because it is later used by the cloud server to determine the aggregation weight of each gateway during weighted aggregation.

After sending the local payload, the gateway waits for the cloud server to generate a new global model. Once a newer global model is available, the gateway retrieves the latest model and checks whether the current global round has reached the predefined maximum round R. If the global round has reached R, the gateway terminates the training process. Otherwise, the gateway uses the latest global model as the reference model for the next local training iteration. Through this process, each gateway repeatedly contributes local learning results to the global intrusion detection model without exposing its raw traffic data.

TAIPEI TECH

3.2.2 Controller Layer Operation Process

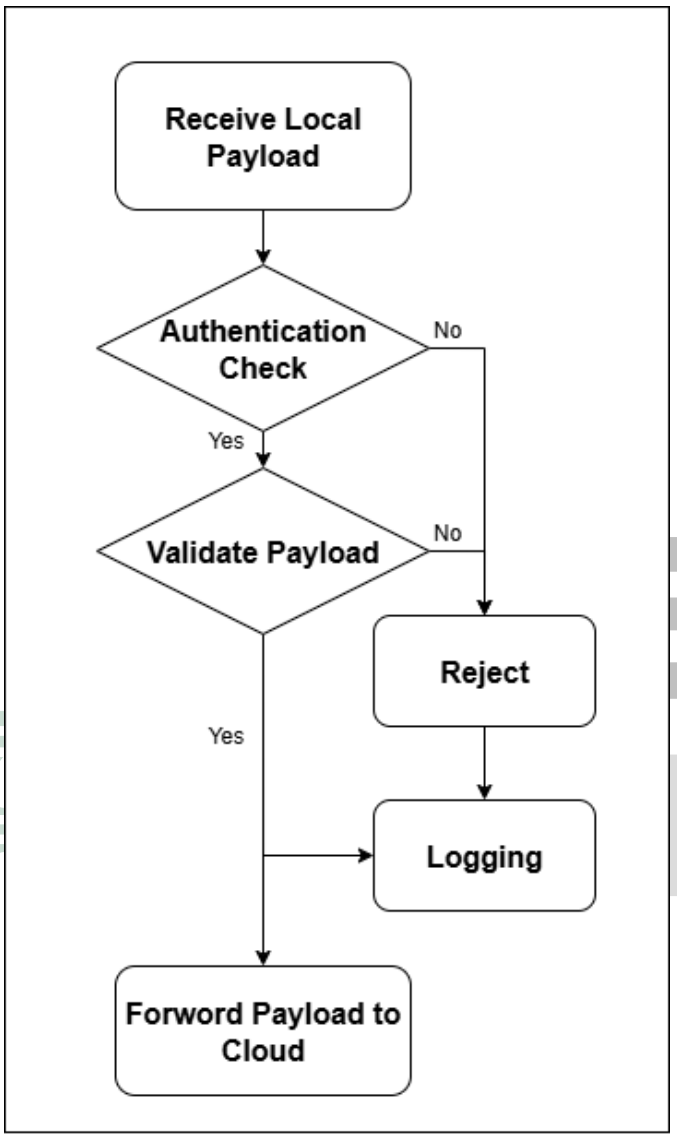


Figure 3.8 Controller workflow.

As shown in Fig. 3.8, the Controller Layer serves as an intermediate validation and security coordination layer between the gateways and the cloud server. Since the proposed KAC-FL framework operates in a distributed IoT environment where gateways may communicate asynchronously, it is necessary to ensure that only legitimate and structurally valid training contributions are allowed to enter the aggregation pipeline. Therefore, the controller performs authentication, payload validation, logging, and request forwarding.

When a local payload is received from a gateway, the controller first performs an authenti-

cation check to verify the legitimacy of the requesting gateway. This authentication mechanism is designed to prevent unauthorized or unknown nodes from injecting malicious or fabricated training contributions into the federated learning process. If the authentication check fails, the request is immediately rejected and recorded in the system log.

For authenticated requests, the controller proceeds to the payload validation stage. This validation process ensures that the received payload conforms to the expected structural and logical requirements of the KAC-FL framework. Specifically, the controller verifies whether the payload contains the required model information and metadata fields, including the gateway identifier, model version, originating global round, local training sample size, and other relevant training metadata. In addition, the controller checks whether the payload format is complete and whether critical information is missing or malformed.

If the payload fails validation, the request is rejected to prevent corrupted or invalid model contributions from affecting the aggregation process. The rejection event is also logged for monitoring and auditing purposes. If the payload passes validation, the controller records the request information in the system log and forwards the validated payload to the cloud server for aggregation.

The logging mechanism is applied to both accepted and rejected requests. This design provides traceability for the distributed training process and supports debugging, monitoring, and security auditing. By introducing the Controller Layer as an intermediate validation checkpoint, the KAC-FL framework improves robustness and reduces the risk of invalid or malicious payloads directly influencing global model updates.

TAIPEI TECH

3.2.3 Cloud Layer Operation Process

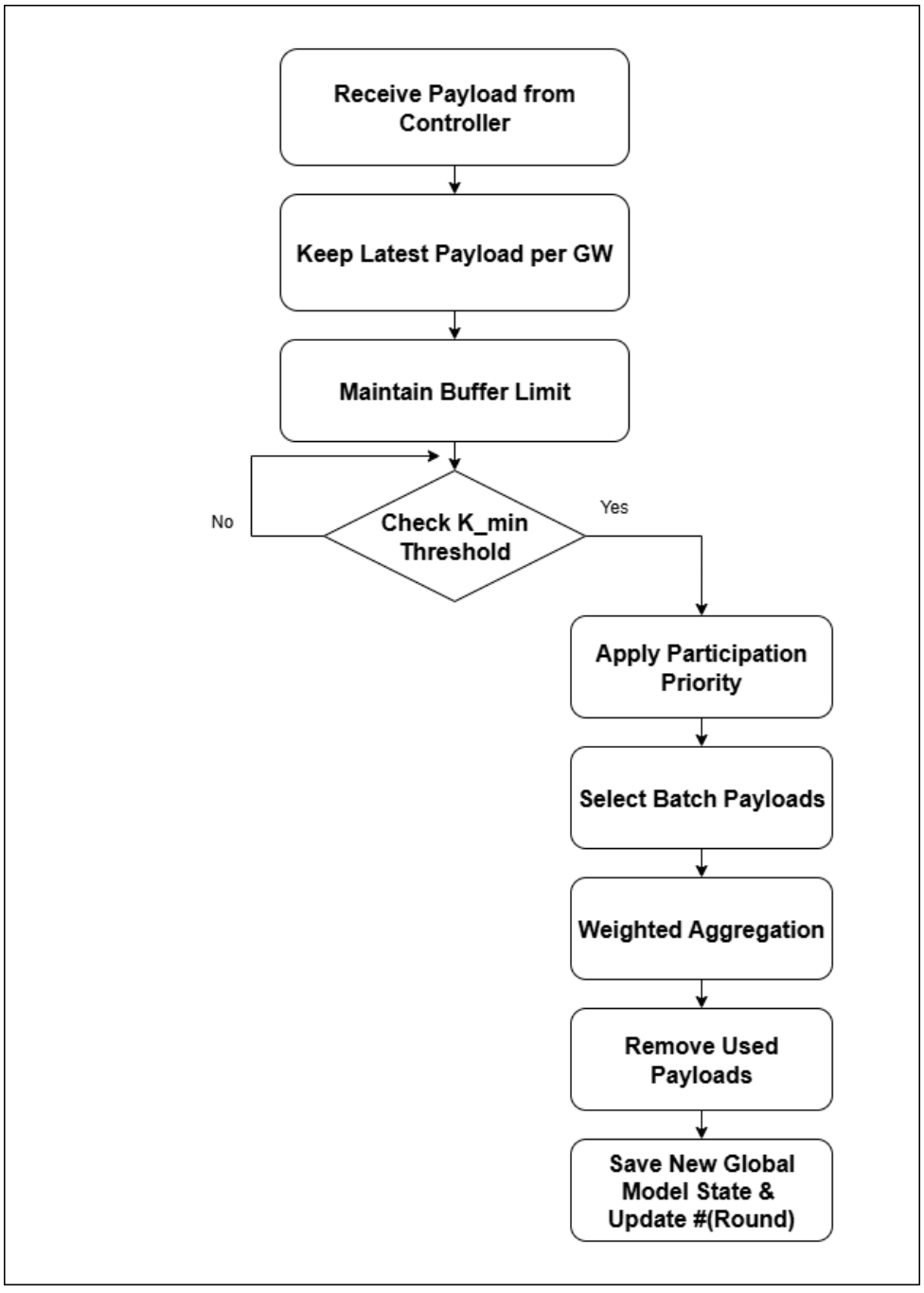


Figure 3.9 Cloud workflow.

As shown in Fig. 3.9, the Cloud Layer serves as the aggregation coordination center of the proposed KAC-FL framework. Its primary responsibilities include payload buffering, aggregation triggering, participation fairness control, weighted model aggregation [3], and global model state management.

When the cloud server receives a validated payload from the Controller Layer, the payload is inserted into the aggregation buffer. To prevent stale contributions from repeatedly occupying storage resources, the cloud maintains only the latest payload from each gateway. If a gateway submits a newer payload before aggregation is triggered, the previously stored payload from the same gateway is replaced. This ensures that the aggregation process always considers the most recent local model contribution from each participating gateway.

To maintain controlled resource usage and stable system behavior, the cloud applies a fixed buffer management policy. The buffer capacity is constrained according to the total number of participating gateways, ensuring that payload accumulation remains bounded even under asynchronous communication conditions.

After buffer insertion, the cloud evaluates whether the aggregation trigger condition has been satisfied. Aggregation is initiated only when the number of distinct gateways with available payloads reaches the predefined aggregation threshold K-min, which is defined as:

$$|S| \geq K_{min} \tag{3.1}$$

where S denotes the set of distinct gateways with available local payloads, and K-min represents the minimum aggregation threshold.

This mechanism allows KAC-FL to provide a controllable trade-off between aggregation responsiveness and training stability. Smaller K-min values enable faster aggregation with lower waiting time, while larger K-min values improve aggregation consistency by incorporating more gateway contributions before updating the global model.

To mitigate participation imbalance in asynchronous federated learning environments, the proposed framework further introduces a participation priority mechanism. In conventional asynchronous settings, faster gateways may repeatedly dominate the aggregation process, causing slower gateways to be rarely selected. To address this issue, the cloud activates the participation priority mechanism when the current aggregation round reaches the late training stage, defined as:

TAIPEI TECH

$$r \geq R - \left\lfloor \frac{N}{K_{min}} \right\rfloor \tag{3.2}$$

where r represents the current aggregation round, R denotes the total predefined global training rounds, and N is the total number of participating gateways.

Once this condition is satisfied, the cloud checks whether any gateway has not yet participated in previous aggregations. If such gateways exist, their payloads are temporarily prioritized for inclusion in the next aggregation round. This mechanism improves fairness among heterogeneous gateways and reduces the risk of persistent exclusion caused by asynchronous performance differences.

The proposed participation priority mechanism is designed to reduce the possibility that slower gateways remain continuously excluded from global aggregation under heterogeneous execution conditions. Instead of enforcing identical participation frequencies for all gateways, the mechanism provides additional aggregation opportunities for gateways that have not yet participated in previous aggregation rounds. Consequently, the proposed mechanism aims to improve participation coverage and reduce persistent gateway exclusion while preserving the asynchronous characteristics of the KAC-FL framework.

After the aggregation condition is satisfied and the participation priority mechanism is applied, the Cloud selects the eligible local model packages for aggregation. In the proposed implementation, each gateway trains an XGBoost model locally and encapsulates the model information required for global aggregation into the local model contribution $L_i^{(r)}$. Therefore, $L_i^{(r)}$ denotes the numerical representation of the locally trained XGBoost model used by the aggregation procedure, rather than the raw training data collected at the gateway.

A data-size-based weighted aggregation strategy is then applied to combine the selected local model contributions. The contribution of each gateway is proportional to the number of samples used for its local training. The updated global model is computed as:

$$G(r+1) \triangleq \frac{\sum_{i \in S} n_i L_i(r)}{\sum_{i \in S} n_i} \tag{3.3}$$

where $G^{(r+1)}$ denotes the updated global model, $L_i^{(r)}$ represents the local XGBoost model

contribution generated by gateway i at round r, and n_i denotes the number of training samples used by gateway i. The aggregation is therefore performed on the model information contained in the local model contributions rather than on raw gateway traffic. Gateways with larger local datasets receive proportionally larger aggregation weights, allowing their locally learned information to contribute more strongly to the resulting global model.

After aggregation, the payloads used in the current round are removed from the buffer to prevent repeated reuse. The newly generated global model is then stored as the updated global model state, which can subsequently be retrieved by gateways during the next iterative training cycle.

In summary, the proposed KAC-FL framework establishes a structured federated intrusion detection workflow through the coordinated interaction of the Gateway, Controller, and Cloud layers. The Gateway Layer is responsible for local data processing and model training while preserving data privacy by keeping raw traffic data at the edge. The Controller Layer serves as an intermediate security checkpoint that ensures only authenticated and structurally valid payloads are allowed to enter the aggregation pipeline. The Cloud Layer incorporates K-min aggregation mechanism into the proposed framework, integrates the participation priority strategy to improve fairness among heterogeneous gateways, and applies weighted aggregation to generate updated global models.

Through this layered operational design, KAC-FL effectively balances training efficiency, aggregation stability, and participation fairness in heterogeneous IoT environments. Compared with conventional synchronous federated learning, the proposed framework reduces unnecessary waiting time caused by slower gateways, while avoiding the instability commonly observed in fully asynchronous aggregation. The operational workflow described in this section establishes the methodological foundation for the practical implementation and deployment of the proposed KAC-FL framework, which will be detailed in the next chapter.

TAIPEI TECH

Chapter 4 Implementation

4.1 Experimental Environment

This section describes the experimental environment used to implement and evaluate the proposed KAC-FL framework, including hardware infrastructure and software configuration.

4.1.1 Hardware Deployment Configuration

The proposed KAC-FL framework was deployed on a desktop host machine running Windows 11, equipped with an Intel Core i7-13700 processor, 32 GB of RAM, and Intel UHD Graphics 770. VMware Workstation was used as the virtualization platform to construct the distributed experimental environment.

To emulate the distributed architecture of the proposed framework, three dedicated virtual machines were configured to represent the Gateway, Controller, and Cloud components. The Gateway virtual machine was further designed to simulate ten logical gateway processes, enabling asynchronous local model training and distributed payload transmission to mimic heterogeneous IoT participants.

Table 4.1 Hardware deployment configuration of the experimental environment.

Virtual Machine	CPU	RAM	Storage
Gateway VM	4 Cores	8 GB	40 GB
Controller VM	1 Core	2 GB	20 GB
Cloud VM	3 Cores	6 GB	30 GB

Deployment Assumption: In this study, the Gateway VM is used to emulate an edge computing gateway rather than a conventional network forwarding gateway. It is assumed to possess sufficient computational resources to perform local data preprocessing and XGBoost-based local model training for multiple connected IoT devices. This deployment reflects an edge computing scenario in which IoT devices are responsible only for sensing and generating

network traffic, while the gateway provides edge intelligence by executing local model training. The Controller VM and Cloud VM are responsible for authentication, aggregation control, and global model management, respectively.

4.1.2 Software Environment Configuration

All virtual machines were configured with Ubuntu 22.04 as the operating system and Python 3.10 as the primary development environment to ensure software consistency across distributed components.

The proposed framework was implemented using Python-based services. The Gateway component employed XGBoost as the local machine learning model for intrusion detection training, while the Controller component utilized Flask to provide RESTful API services for authentication, payload validation, and request forwarding. Communication among distributed components was conducted through RESTful API interactions over a VMware NAT-based virtual network.

Table 4.2 Software environment configuration of the experimental environment.

Component	**Configuration**
Host Operating System	Windows 11 [22]
Virtualization Platform	VMware Workstation [23]
Virtual Machine OS	Ubuntu 22.04 [24]
Programming Language	Python 3.10 [25]
Machine Learning Framework	XGBoost [9]
Web Framework	Flask [26]
Communication Protocol	RESTful API [27]
Network Configuration	VMware NAT-based Virtual Network

TAIPEI TECH

4.2 Implementation Details

This section summarizes the implementation details of each distributed component in the proposed KAC-FL framework, including their functional responsibilities and software dependencies.

- **Gateway VM**

  The Gateway VM was configured to simulate ten logical gateway processes representing distributed IoT participants. Its primary responsibilities include loading assigned local dataset partitions, performing preprocessing and sampling, training local intrusion detection models using XGBoost, serializing local model payloads, transmitting payloads to the Controller VM, and fetching updated global models for subsequent training rounds.

Table 4.3 Gateway VM software dependencies.

Package	**Version**	**Purpose**	**License**
Python	3.10	Runtime environment	PSF License
XGBoost	3.2.0	Local model training	Apache 2.0
Pandas [28]	2.3.3	Data preprocessing	BSD 3-Clause
NumPy [29]	2.2.6	Numerical computation	BSD 3-Clause
Scikit-learn [30]	1.7.2	Preprocessing support	BSD 3-Clause
Requests [31]	2.33.1	REST API communication	Apache 2.0

- **Controller VM**

  The Controller VM serves as an intermediate coordination and validation node between the Gateway and Cloud components. Its implementation includes receiving local model payloads, performing authentication verification, validating payload structure and metadata, logging request outcomes, and forwarding validated payloads to the Cloud VM.

Table 4.4 Controller VM software dependencies.

Package	Version	Purpose	License
Python	3.10	Runtime environment	PSF License
Flask	3.1.3	REST API service	BSD 3-Clause
Requests	2.33.1	Cloud communication	Apache 2.0
Blinker [32]	1.9.0	Flask signal support	MIT License
Werkzeug [33]	3.1.8	WSGI utility support	BSD 3-Clause
Jinja2 [34]	3.1.6	Template engine support	BSD 3-Clause

- **Cloud VM**

  The Cloud VM acts as the aggregation coordinator of the proposed KAC-FL framework. Its implementation includes receiving validated payloads, maintaining the aggregation buffer, retaining the latest payload from each gateway, triggering aggregation based on the K-min threshold, applying the participation priority mechanism, performing weighted model aggregation, and updating the global model state.

Table 4.5 Cloud VM software dependencies.

Package	Version	Purpose	License
Python	3.10	Runtime environment	PSF License
Flask	3.1.3	Global aggregation service	BSD 3-Clause
XGBoost	3.2.0	Global model management	Apache 2.0
NumPy	2.2.6	Weighted aggregation computation	BSD 3-Clause
Pandas	2.3.3	Experimental data handling	BSD 3-Clause
OpenPyXL [35]	3.1.5	Result export support	MIT License

4.3 Dataset Preparation and Non-IID Distribution

The CICIDS2017 dataset was adopted as the benchmark dataset for evaluating the proposed KAC-FL framework. Following the preprocessing procedures described in the previous chap-

ter, the processed dataset was partitioned into global evaluation data and gateway-local datasets using a deliberately designed non-IID allocation strategy to simulate heterogeneous IoT deployment conditions.

4.3.1 Dataset Split

To ensure fair evaluation across different aggregation settings, the processed dataset was divided into a fixed global test set and gateway-local datasets. The global test set was reserved exclusively for evaluating the aggregated global intrusion detection model, ensuring consistent performance comparison throughout all experiments. Each gateway dataset was further divided into local training and validation subsets for local model development.

Table 4.6 Dataset split configuration.

Level	**Purpose**	**Samples**
Global Test Set	Evaluate global IDS model	200,000
Gateway Dataset	Local dataset per gateway	50,000
Gateway Training Set	Local model training	40,000 (80%)
Gateway Validation Set	Local model validation	10,000 (20%)

4.3.2 Gateway Non-IID Allocation

To emulate heterogeneous IoT environments, a deliberately non-IID data allocation strategy was applied across the ten gateway nodes. Rather than distributing data uniformly, each gateway was assigned a distinct benign-to-attack ratio, dominant attack category, and representative deployment scenario to reflect diverse operational conditions commonly observed in practical distributed networks.

Gateways GW1 and GW2 were configured with traffic distributions dominated by benign traffic, representing relatively stable enterprise and semi-protected network environments. Gateways GW3 through GW9 were assigned attack-heavy traffic distributions, each emphasizing a different primary attack type to simulate highly exposed or compromised service environments.

GW10 was configured with a balanced traffic composition to represent mixed remote access scenarios.

This allocation introduces significant statistical heterogeneity among participating gateways, enabling realistic evaluation of the proposed asynchronous federated intrusion detection framework.

Table 4.7 Non-IID data allocation across gateway nodes.

Gateway	Benign	Attack	Main Attack Type	Simulated Scenario
GW1	45,000	5,000	PortScan / Patator	Office VLAN
GW2	35,000	15,000	PortScan	DMZ
GW3	10,000	40,000	DDoS	Public Service
GW4	10,000	40,000	DoS Family	Internal Service
GW5	10,000	40,000	Web BruteForce	Login Portal
GW6	10,000	40,000	XSS	Web Application
GW7	10,000	40,000	SQL Injection	Database Web Service
GW8	10,000	40,000	Infiltration	Compromised Host
GW9	10,000	40,000	Bot	IoT Botnet
GW10	25,000	25,000	Patator	Remote Access

4.4 Experimental Configuration

To evaluate the proposed KAC-FL framework under heterogeneous IoT-inspired environments, a series of controlled experiments were conducted using different aggregation threshold settings. The experimental configuration was designed to compare fully asynchronous, intermediate, and near-synchronous aggregation behaviors under a consistent deployment environment.

Table 4.8 Experimental parameter configuration.

Parameter	**Configuration**
Number of Gateway Nodes	10
Global Training Rounds	20
Aggregation Threshold	$K_{min} = 1 \sim 10$
Gateway Local Training Samples	40,000
Gateway Validation Samples	10,000
Global Test Samples	200,000
Local Learning Model	XGBoost
Deployment Environment	VMware-based distributed virtual environment

To investigate the impact of aggregation synchronization levels, the aggregation threshold K-min was varied from 1 to 10. When $K_{\text{min}} = 1$, aggregation is triggered immediately upon receiving the first eligible gateway payload, representing a fully asynchronous setting. When $K_{\text{min}} = 10$, aggregation occurs only after all gateway contributions are available, corresponding to conventional synchronous federated learning. Intermediate threshold values ($K_{\text{min}} = 2 \sim 9$) represent partially asynchronous aggregation configurations, enabling analysis of the trade-off between responsiveness and aggregation stability.

Each experimental configuration was executed under the same dataset allocation and deployment environment to ensure fair comparison across different synchronization levels.

Gateway-Specific Training Delay To emulate heterogeneous edge computing environments, gateway-specific artificial training delays were introduced during the local training process. Instead of assigning identical execution times to all gateways, different fixed delay values were assigned to individual gateways to represent variations in computational capability and processing speed among edge gateways. Specifically, the delays were configured as follows: GW1–GW4 were assigned a delay of 1 second, GW5–GW6 a delay of 2 seconds, GW7 a delay of 8 seconds, GW8 a delay of 40 seconds, GW9 a delay of 42 seconds, and GW10 a delay of 50 seconds. These delay values remained unchanged throughout each experimental execution.

The introduced delays affect only the transmission time of local model packages and do not modify the local datasets, XGBoost training procedure, or model parameters. Consequently, asynchronous model arrivals are generated naturally, allowing the proposed K-min aggregation control mechanism and participation priority mechanism to be evaluated under heterogeneous gateway execution conditions.

The delay values were intentionally configured to create noticeable execution heterogeneity among gateways, thereby emulating edge devices with different computational capabilities and network response times. The selected values are experimental settings rather than measurements obtained from specific hardware platforms.

4.5 Implementation Steps

This section presents the practical implementation workflow used to execute the proposed KAC-FL framework. To ensure reproducibility and consistency across different aggregation threshold settings, all experiments followed the same structured implementation procedure, which is divided into three phases: environment initialization, distributed training execution, and experimental repetition with result collection.

4.5.1 Environment Initialization

The experimental environment was initialized as follows.

Deployment: Three virtual machines were deployed using VMware Workstation to represent the Gateway, Controller, and Cloud components. The Gateway VM executed ten independent logical gateway processes to emulate heterogeneous edge gateways.

Software Configuration: Ubuntu 22.04 and Python 3.10 were installed on all virtual machines. The required Python packages, including Flask, XGBoost, pandas, NumPy, scikit-learn, and Requests, were installed.

Network Configuration: The three virtual machines were connected through a VMware NAT

network. Static IP addresses, service ports, Cloud endpoints, and authentication keys were configured.

Dataset Preparation: The CICIDS2017 dataset was preprocessed and partitioned into ten heterogeneous gateway datasets together with one fixed global testing dataset.

Service Initialization: The Cloud service, Controller service, and ten logical gateway processes were started sequentially to initialize the distributed KAC-FL framework.

4.5.2 Distributed Training Execution

After completing the environment initialization, the distributed training process of the proposed KAC-FL framework is executed. The Controller and Cloud services are first started, followed by the initialization of the global model. Ten logical gateway processes are then launched to simulate distributed IoT edge gateways. Each gateway retrieves the latest global model, trains a local XGBoost model using its assigned Non-IID dataset, simulates gateway-specific training delay, and generates a local model package.

The generated local model package is transmitted to the Controller, where gateway authentication, package validation, transmission logging, and forwarding are performed. Valid packages are forwarded to the Cloud server and stored in the aggregation buffer. Only the latest package from each gateway is retained in the buffer.

During global training, the Cloud continuously checks whether the number of available local model packages satisfies the predefined aggregation threshold K-min. When the aggregation condition is satisfied, the participation priority mechanism is applied to improve gateway participation fairness. The Cloud then selects the aggregation set and performs weighted aggregation to generate the updated global model. After aggregation, the used local model packages are removed from the buffer, the updated global model is stored, and the latest global model is broadcast to all gateways for the next communication round.

Algorithm 1 presents the distributed local training procedure, while Algorithm 2 summarizes the global aggregation procedure.

Algorithm 1 Distributed Local Training Procedure

Start Controller and Cloud services
Initialize global model $G^{(0)}$
Launch ten logical gateway processes
for all gateway i **do**
    Retrieve latest global model
    Train local XGBoost model L_i
    Simulate gateway-specific training delay
    Generate local model package
    Send local model package to Controller
    Authenticate gateway identity
    Validate received package
    Record transmission log
    Forward valid package to Cloud
    Store package into aggregation buffer
end for

TAIPEI TECH

Algorithm 2 Global Aggregation Procedure

while global round $r < R$ **do**
  if $|B| \geq K_{\text{min}}$ **then**
    Apply participation priority mechanism
    Select aggregation set S_r
    Perform weighted aggregation
    Generate global model $G^{(r+1)}$
    Remove aggregated packages from buffer
    Save updated global model
    Broadcast $G^{(r+1)}$ to all gateways
    $r \leftarrow r + 1$
  else
    Wait for additional local model packages
  end if
end while

TAIPEI TECH

The distributed training process begins by initializing the Controller, Cloud server, and the initial global model. Ten gateway processes are then launched, each independently training a local XGBoost model using its assigned Non-IID dataset. After completing local training, each gateway encapsulates the trained model and its associated metadata into a local model package and transmits it to the Controller.

The Controller is responsible for authenticating the gateway, validating the received package, recording communication logs, and forwarding valid packages to the Cloud server. The Cloud stores incoming packages in the aggregation buffer while retaining only the most recent package from each gateway.

Once the number of buffered packages reaches the predefined aggregation threshold K-min, the Cloud first applies the participation priority mechanism to improve gateway participation fairness, and then selects the aggregation set for global aggregation. The global model is generated using the weighted aggregation strategy described in Chapter 3. After aggregation, the Cloud removes the utilized local model packages from the buffer, stores the updated global

model, broadcasts the latest model to all gateways, and proceeds to the next communication round until all training rounds are completed.

4.5.2.1 Experimental Execution and Result Collection

To compare the influence of different aggregation thresholds, the complete distributed training workflow was executed separately for each K_{min} configuration from $K_{\text{min}} = 1$ to $K_{\text{min}} = 10$. Each configuration used the same gateway-local datasets, global testing dataset, model parameters, number of gateways, and total number of global rounds to maintain consistent experimental conditions. Before executing each configuration, the global model, aggregation buffer, round counter, model version, and gateway participation records were reset to their initial states.

For each K_{min} configuration, one complete end-to-end training execution was conducted. After the predefined number of global rounds was completed, the final global model was evaluated using the fixed global testing dataset. The corresponding F1-score, Recall, and AUC were recorded together with the total runtime, aggregation time, and gateway participation information. These records were then exported for comparative analysis in Chapter 5.

Algorithm 3 summarizes the experimental execution and result collection procedure.

TAIPEI TECH

Algorithm 3 Experimental Execution and Result Collection

1: Set the total number of global rounds to $R \leftarrow 20$
2: Set the gateway count to $N \leftarrow 10$
3: Load the fixed gateway-local datasets $\{D_1, D_2, \ldots, D_N\}$
4: Load the fixed global testing dataset D_{test}
5: Initialize the experimental result table $T \leftarrow \emptyset$
6: **for** $K_{\text{min}} \leftarrow 1$ to 10 **do**
7: Restore the initial global model $G^{(0)}$
8: Clear the aggregation buffer B
9: Reset the global round counter $r \leftarrow 0$
10: Reset the global model version
11: Reset all gateway participation records
12: Restart the Gateway, Controller, and Cloud services
13: Start the experiment timer
14: Execute the complete distributed training workflow
15: Stop the experiment timer
16: Obtain the final global model $G^{(R)}$
17: Evaluate $G^{(R)}$ using D_{test}
18: Calculate F1-score, Recall, and AUC
19: Record the total runtime and aggregation time
20: Record the participation count of each gateway
21: Save the round-level and final experimental results
22: Append the results for the current K_{min} to T
23: **end for**
24: Export T as the experimental summary CSV file
25: Generate the comparison figures for Chapter 5

Algorithm 3 describes the procedure used to evaluate the different aggregation threshold configurations under the same experimental setup. Lines 1–5 initialize the common experimental parameters and datasets. Lines 6–22 execute one complete training process for each K_{min}

TAIPEI TECH

configuration from 1 to 10. Before each execution, all model states, aggregation records, and service components are reset to ensure that each configuration starts from the same initial condition.

After each complete training execution, the final global model is evaluated using the same global testing dataset, and the corresponding detection performance and execution efficiency metrics are recorded. The collected results are then exported to a summary file and used to generate the comparison figures presented in Chapter 5.

Since each $K_{\min}$ configuration is evaluated using one complete execution under controlled and identical experimental conditions, the reported values are interpreted as representative observations for comparing the behavioral differences among aggregation threshold settings rather than as statistical estimates derived from repeated independent trials.

In summary, this chapter presented the practical implementation of the proposed KAC-FL framework, including the experimental environment, system deployment configuration, implementation details of distributed components, dataset preparation, and experimental settings. These implementation and experimental configurations establish a consistent and reproducible evaluation environment for assessing the effectiveness of the proposed asynchronous federated intrusion detection framework under heterogeneous IoT-inspired conditions. The following chapter presents the experimental results and provides a detailed performance analysis of the proposed framework under different aggregation threshold settings.

TAIPEI TECH

Chapter 5 Results and Analysis

This chapter presents the experimental results and performance analysis of the proposed KAC-FL framework under heterogeneous IoT-inspired environments. Based on the implementation setup and experimental configurations described in Chapter 4, a series of controlled experiments were conducted to evaluate the effectiveness of the proposed asynchronous federated intrusion detection framework under different aggregation threshold settings.

To provide a comprehensive evaluation, both detection performance and system efficiency are analyzed from multiple perspectives. Detection capability is assessed using standard classification metrics, while execution efficiency is evaluated through runtime-related measurements under varying synchronization levels. In addition, comparative analysis is conducted to examine the trade-off between asynchronous responsiveness, aggregation stability, and overall detection effectiveness.

The experimental results presented in this chapter aim to demonstrate whether the proposed KAC-FL framework can achieve a practical balance between intrusion detection performance and training efficiency in heterogeneous distributed environments.

5.1 Evaluation Metrics

To quantitatively assess the performance of the proposed KAC-FL framework, this study adopts commonly used evaluation metrics [36] for both intrusion detection effectiveness and system execution efficiency. Detection performance is evaluated using precision, recall, F1-score, and Area Under the ROC Curve (AUC), while system efficiency is assessed using average aggregation time and total runtime. These metrics provide a comprehensive evaluation of the proposed asynchronous federated intrusion detection framework from both classification and operational perspectives.

In binary intrusion detection tasks, classification outcomes are categorized into four fundamental cases: true positive (TP), true negative (TN), false positive (FP), and false negative (FN). A true positive indicates that malicious traffic is correctly identified as an attack, while a true

negative indicates that benign traffic is correctly classified as normal. A false positive represents normal traffic incorrectly predicted as an attack, whereas a false negative represents malicious traffic that is mistakenly classified as benign. Based on these definitions, the evaluation metrics adopted in this study are described as follows:

1. **Precision:** Precision measures the proportion of correctly predicted attack instances among all samples classified as attacks. It reflects the reliability of positive attack predictions and indicates how often the model's attack alarms are correct.

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

A higher precision indicates fewer false positive predictions, reducing unnecessary security alerts and administrative overhead.

2. **Recall:** Recall measures the proportion of actual attack instances that are correctly identified by the intrusion detection model. Since false negatives may allow malicious traffic to bypass detection, recall is particularly important in cybersecurity applications.

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

A higher recall indicates stronger attack detection capability and lower risk of undetected threats.

3. **F1-score:** F1-score summarizes the overall detection performance by balancing precision and recall through their harmonic mean. It is especially suitable for intrusion detection scenarios where class imbalance may exist.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5.3}$$

A higher F1-score indicates better overall classification effectiveness by jointly considering both false positives and false negatives.

4. **Area Under the ROC Curve (AUC):** AUC evaluates the model's ability to distinguish between benign and malicious traffic across different classification thresholds. Unlike threshold-dependent metrics, AUC reflects the classifier's overall discrimination capability.

$$AUC = \int_0^1 TPR(FPR)\, d(FPR) \tag{5.4}$$

where TPR denotes the true positive rate and FPR denotes the false positive rate. A higher AUC indicates stronger discrimination capability between attack and benign traffic.

5. **Average Aggregation Time:** Average aggregation time measures the average computational time required for each global aggregation operation during federated training. This metric reflects the aggregation overhead introduced under different synchronization thresholds.

$$AvgAggregationTime = \frac{\sum_{r=1}^{R} T_{agg}^{(r)}}{R} \tag{5.5}$$

where $T_{agg}^{(r)}$ denotes the aggregation time in global round r, and R represents the total number of completed global rounds.

6. **Total Runtime:** Total runtime represents the total execution time required to complete the entire federated training process from initialization to final model generation.

$$TotalRuntime = T_{end} - T_{start} \tag{5.6}$$

This metric is used to evaluate the overall efficiency of the proposed framework under different aggregation threshold configurations.

TAIPEI TECH

5.2 Detection Performance Analysis

This section analyzes the intrusion detection performance of the proposed KAC-FL framework under different aggregation threshold settings. To evaluate the impact of different aggregation threshold settings on detection effectiveness, experiments were conducted with K-min values ranging from 1 to 10. Detection performance is assessed using F1-score, recall, and AUC.

5.2.1 F1-score Analysis

Figure 5.1 presents the relationship between aggregation threshold settings and final average test F1-score.

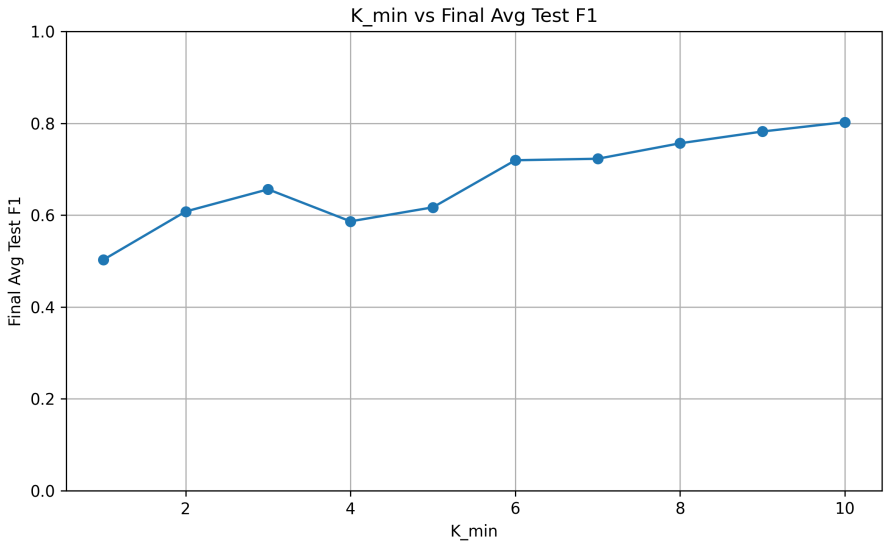


Figure 5.1 Impact of $K_{\min}$ on final average test F1-score.

As shown in Figure 5.1, the F1-score generally improves as the aggregation threshold increases. When $K_{\min} = 1$, corresponding to the fully asynchronous setting, the framework achieves the lowest F1-score, approximately 0.50. This indicates that overly aggressive asynchronous aggregation may reduce model stability and negatively affect overall classification effectiveness.

As K-min increases, the F1-score improves steadily, reaching approximately 0.80 when $K_{\text{min}} = 10$, which represents the near-synchronous configuration. This trend suggests that incorporating more gateway contributions before aggregation improves global model quality by reducing the influence of incomplete or unstable local updates.

Notably, intermediate threshold settings such as $K_{\text{min}} = 3$ to $K_{\text{min}} = 6$ already achieve substantial performance improvement compared with the fully asynchronous baseline, indicating that the proposed framework can maintain acceptable detection effectiveness without requiring full synchronization.

5.2.2 Recall Analysis

Figure 5.2 illustrates the impact of different aggregation thresholds on attack detection recall.

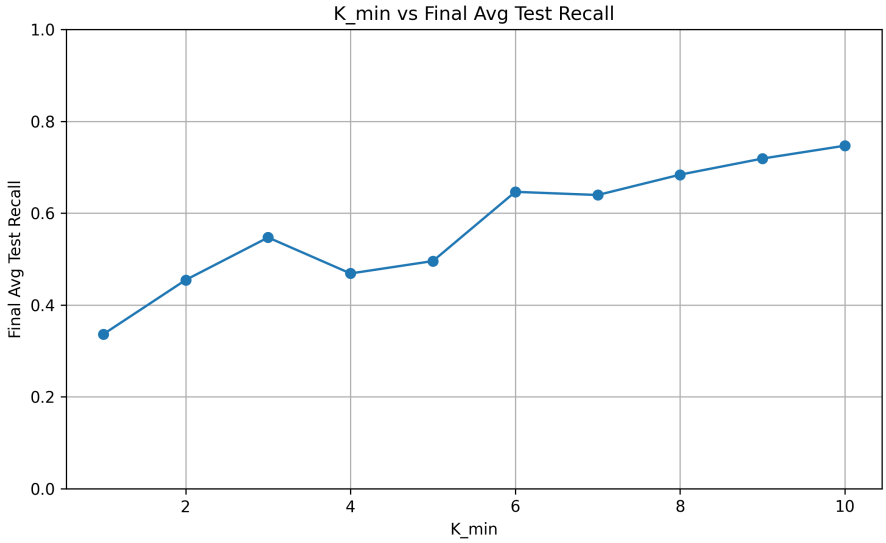


Figure 5.2 Impact of K_{min} on final average test recall.

As shown in Figure 5.2, recall increases significantly as the aggregation threshold becomes larger. Under the fully asynchronous setting ($K_{\text{min}} = 1$), recall remains relatively low at approximately 0.33, indicating that a considerable number of attack instances are not successfully detected.

As synchronization increases, recall improves progressively, reaching approximately 0.75 at $K_{\text{min}} = 10$. This suggests that aggregating a larger number of gateway contributions improves the model's ability to detect malicious traffic more reliably.

Compared with F1-score, recall exhibits a more noticeable performance gap between asynchronous and near-synchronous configurations. This indicates that insufficient synchronization primarily affects attack detection completeness rather than overall classification balance, which is particularly important in cybersecurity applications where false negatives are highly undesirable.

5.2.3 AUC Analysis

Figure 5.3 presents the AUC performance under different aggregation threshold settings.

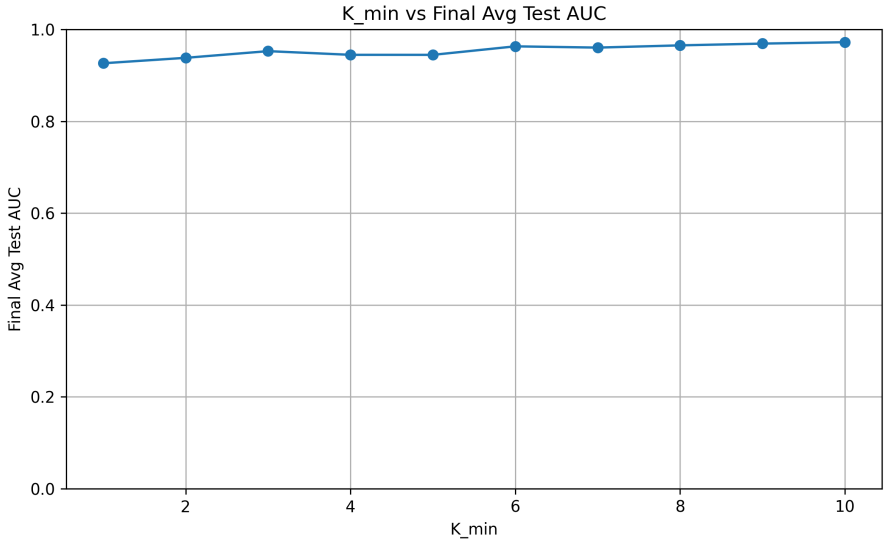


Figure 5.3 Impact of K_{min} on final average test AUC.

Unlike F1-score and recall, AUC remains consistently high across all experimental configurations, ranging approximately from 0.92 to 0.97. This indicates that the proposed framework maintains strong discrimination capability between benign and malicious traffic regardless of synchronization level.

Although a gradual improvement is still observable as K-min increases, the performance gap is relatively small compared with other detection metrics. This suggests that even un-

der asynchronous conditions, the global model retains effective classification boundary learning, while synchronization primarily improves prediction stability and attack detection completeness.

Overall, the detection performance analysis demonstrates that increasing aggregation synchronization improves detection effectiveness, particularly in terms of F1-score and recall. However, intermediate aggregation thresholds already provide competitive detection performance without requiring the full waiting overhead of synchronous aggregation, supporting the practical design objective of the proposed KAC-FL framework.

5.3 Training Efficiency Analysis

This section evaluates the execution efficiency of the proposed KAC-FL framework under different aggregation threshold settings. Since asynchronous federated learning is primarily designed to reduce synchronization overhead and improve training responsiveness, runtime-related metrics are analyzed to examine how different K-min configurations affect overall execution efficiency.

5.3.1 Total Runtime Analysis

Total runtime is used to measure the complete execution time required to finish the entire federated training process, including local training, payload transmission, synchronization waiting, aggregation, and global model updates.

Figure 5.4 presents the relationship between aggregation threshold settings and total runtime.

TAIPEI TECH

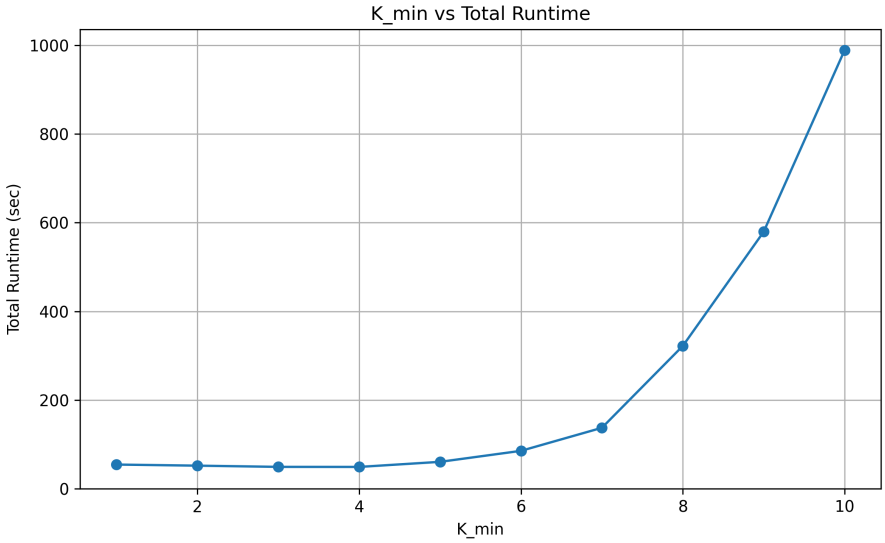


Figure 5.4 Impact of K_{min} on total runtime.

As shown in Figure 5.4, total runtime remains relatively low and stable when the aggregation threshold is configured between $K_{\text{min}} = 1$ and $K_{\text{min}} = 4$. In this range, the framework benefits from asynchronous aggregation behavior, allowing global updates to be triggered without excessive waiting for delayed gateway contributions. This results in fast overall execution with minimal synchronization overhead.

However, as the aggregation threshold increases beyond $K_{\text{min}} = 5$, total runtime begins to increase noticeably. This indicates that requiring more gateway payloads before triggering aggregation introduces additional waiting time, reducing the responsiveness advantage of asynchronous federated learning.

The runtime increase becomes particularly significant under high synchronization settings. When $K_{\text{min}} = 8$ to $K_{\text{min}} = 10$, total runtime grows sharply, reaching nearly 1000 seconds under the near-synchronous configuration. In contrast, intermediate asynchronous configurations such as $K_{\text{min}} = 3$ complete the full training process in approximately 50 seconds.

This substantial performance gap demonstrates that higher synchronization thresholds introduce considerable execution overhead due to waiting for slower gateway participants. While increased synchronization improves model consistency, excessive waiting significantly reduces the practical efficiency of distributed federated intrusion detection.

5.3.2 Aggregation Time Analysis

To further analyze the source of execution overhead, the average aggregation computation time is examined separately. This metric measures only the time required for the Cloud layer to perform the weighted aggregation operation after sufficient gateway payloads have been collected.

Figure 5.5 illustrates the average aggregation time under different aggregation threshold settings.

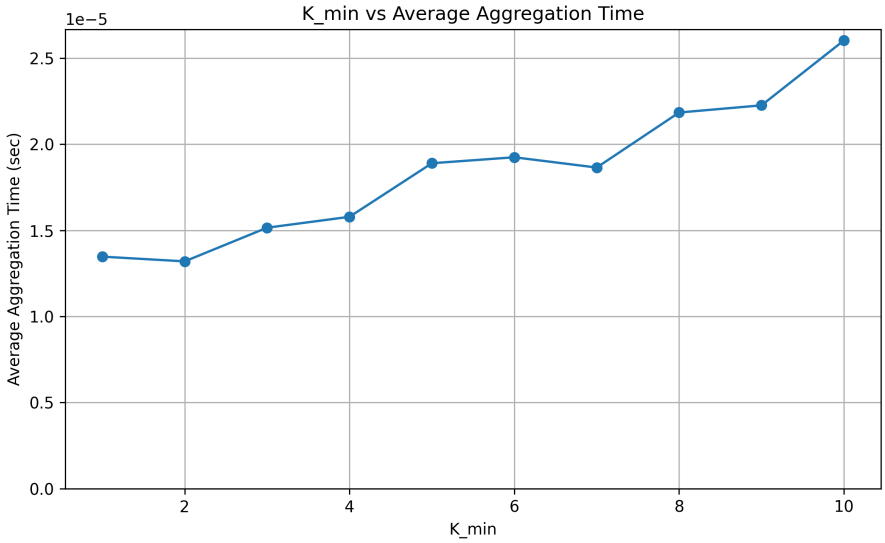


Figure 5.5 Impact of $K_{\min}$ on average aggregation time.

As shown in Figure 5.5, the average aggregation time remains extremely low across all experimental settings, ranging approximately from 1.3×10^{-5} to 2.6×10^{-5} seconds. Although a gradual increase can be observed as K-min becomes larger, the absolute computational overhead remains negligible.

This observation indicates that the weighted aggregation operation itself contributes minimally to overall runtime. Therefore, the dramatic runtime increase observed in Figure 5.4 cannot be attributed to aggregation computation cost.

Instead, the primary source of execution overhead is synchronization waiting, where the system must delay aggregation until the required number of gateway payloads is received. As K-min approaches the fully synchronous setting, this waiting overhead becomes increasingly

dominant.

Overall, these results confirm that the proposed KAC-FL framework preserves the computational efficiency of asynchronous aggregation while allowing configurable synchronization control. This supports the design objective of achieving a practical trade-off between execution efficiency and detection performance.

5.4 Comparative Analysis and Discussion

To provide a clearer understanding of the trade-offs introduced by different synchronization strategies, representative aggregation threshold settings are selected for comparative analysis, including the fully asynchronous baseline ($K_{\text{min}} = 1$), two intermediate KAC-FL configurations ($K_{\text{min}} = 3$ and $K_{\text{min}} = 6$), and the near-synchronous baseline ($K_{\text{min}} = 10$).

Table 5.1 Representative performance comparison under different aggregation threshold settings

Configuration	F1-score	Recall	AUC	Total Runtime (sec)
Fully Async ($K_{\text{min}} = 1$)	0.50	0.33	0.92	55
KAC-FL (Low-overhead, $K_{\text{min}} = 3$)	0.66	0.55	0.95	50
KAC-FL (Balanced, $K_{\text{min}} = 6$)	0.72	0.65	0.96	85
Near-Sync ($K_{\text{min}} = 10$)	0.80	0.75	0.97	990

Based on the experimental results presented in the previous sections, clear trade-offs can be observed between detection performance and execution efficiency under different synchronization configurations. These observations are consistent with previous studies on asynchronous federated learning, which reported a trade-off between synchronization efficiency and model convergence stability.

The fully asynchronous configuration [5] ($K_{\text{min}} = 1$) achieves fast execution by minimizing synchronization waiting overhead. However, this aggressive asynchronous behavior leads to the weakest detection performance among all evaluated configurations, particularly in recall, indicating that a considerable number of malicious traffic instances remain undetected. This suggests that triggering aggregation with only a single gateway contribution may reduce model

stability and weaken intrusion detection effectiveness [17].

Introducing K-min controlled aggregation significantly improves detection performance while preserving asynchronous execution advantages. The low-overhead intermediate configuration ($K_{min} = 3$) improves F1-score, recall, and AUC compared with the fully asynchronous baseline, while maintaining nearly identical runtime performance. This demonstrates that even modest synchronization control can substantially enhance model quality without sacrificing execution efficiency.

As the aggregation threshold increases further, the balanced intermediate configuration ($K_{\text{min}} = 6$) provides additional detection performance improvements, achieving stronger F1-score and recall while still maintaining practical execution time. Compared with the low-overhead configuration, this setting reflects a more performance-oriented synchronization strategy while remaining significantly more efficient than near-synchronous execution.

The near-synchronous configuration ($K_{\text{min}} = 10$) achieves the highest detection performance across all evaluated metrics. However, this improvement comes at the cost of extremely high runtime overhead, as the system must wait for nearly all gateway contributions before each aggregation operation can proceed. The total runtime increases dramatically, making this configuration less practical for time-sensitive distributed intrusion detection scenarios.

Although $K_{\text{min}} = 10$ achieved the highest F1-score and Recall, it also resulted in the longest runtime due to its near-synchronous aggregation behavior. In contrast, $K_{\text{min}} = 1$ and $K_{\text{min}} = 3$ provided faster model updates and shorter execution times, but their detection performance was noticeably lower.

Among all evaluated configurations, $K_{\text{min}} = 6$ achieved the most balanced trade-off between detection effectiveness and training efficiency. Compared with the near-synchronous setting ($K_{\text{min}} = 10$), it substantially reduced total runtime while preserving most of the gains in F1-score, Recall, and AUC. Therefore, $K_{\text{min}} = 6$ can be regarded as the balanced operating point of the proposed KAC-FL framework, providing a practical compromise between synchronization overhead and model performance.

These observations demonstrate that no single synchronization configuration universally dominates across all evaluation criteria. Instead, different K-min settings offer distinct trade-offs

between execution efficiency and detection effectiveness.

The proposed KAC-FL framework addresses this limitation by introducing configurable synchronization control through K-min controlled aggregation. Rather than relying solely on fully asynchronous or near-synchronous aggregation strategies, KAC-FL enables flexible deployment choices according to practical operational requirements. Lower intermediate thresholds favor execution efficiency, while higher intermediate thresholds prioritize stronger detection performance, allowing the framework to adapt to diverse distributed intrusion detection environments.

Although the participation priority mechanism is designed to reduce persistent gateway exclusion under asynchronous training, the objective of this study is to evaluate the impact of different $K_{\min}$ configurations on intrusion detection performance and execution efficiency. Therefore, the discussion in this chapter focuses primarily on the influence of aggregation thresholds, while the participation priority mechanism serves as a supporting control strategy for improving gateway participation opportunities throughout the training process.

5.5 Summary

This chapter presented the experimental results and performance analysis of the proposed KAC-FL framework under heterogeneous IoT-inspired environments. Detection performance evaluation showed that increasing the aggregation threshold generally improves intrusion detection effectiveness, particularly in terms of F1-score and recall, while AUC remained consistently high across all experimental configurations.

Training efficiency analysis revealed that lower aggregation thresholds significantly reduce total runtime by minimizing synchronization waiting overhead, whereas higher synchronization settings introduce substantial execution delays despite improved detection performance. Further analysis confirmed that the computational cost of aggregation itself remains negligible, indicating that synchronization waiting is the primary source of runtime overhead.

Comparative analysis demonstrated that different aggregation threshold settings provide distinct trade-offs between execution efficiency and detection effectiveness. Rather than relying

on fixed fully asynchronous or near-synchronous aggregation strategies, the proposed KAC-FL framework enables configurable synchronization control through K-min controlled aggregation, allowing flexible adaptation to different deployment requirements.

Overall, the experimental results validate the effectiveness of the proposed framework in achieving a practical balance between intrusion detection capability and execution efficiency in distributed federated intrusion detection environments. The next chapter concludes this study and discusses potential directions for future work.

TAIPEI TECH

Chapter 6 Conclusion and Future Work

6.1 Conclusion

This thesis proposed KAC-FL, a federated intrusion detection framework designed for heterogeneous IoT environments. The motivation of this study stems from the limitations of conventional centralized intrusion detection systems, which require raw traffic data to be transmitted to a central server, leading to privacy concerns, communication overhead, and limited scalability in distributed IoT deployments. Although federated learning provides a privacy-preserving alternative by enabling decentralized model training, conventional synchronous federated learning suffers from excessive waiting time caused by slower participants, while fully asynchronous federated learning may introduce instability and unfair participation among heterogeneous nodes.

To address these challenges, this study designed KAC-FL as a practical asynchronous federated intrusion detection framework that balances training efficiency, aggregation stability, and participation fairness. Instead of relying on near-synchronous aggregation, the proposed framework integrates a K-min aggregation strategy to provide controllable aggregation timing.

To further improve fairness in heterogeneous environments, this study introduced a participation priority mechanism that monitors gateway participation during later training stages and prioritizes underrepresented gateways when necessary. This mechanism reduces the long-term exclusion problem commonly observed in asynchronous federated learning, where faster participants may repeatedly dominate the aggregation process.

In addition, a three-layer operational architecture consisting of Gateway, Controller, and Cloud layers was designed to support practical deployment. The Gateway Layer performs local preprocessing and XGBoost-based intrusion detection model training while preserving raw traffic privacy at the edge. The Controller Layer provides intermediate authentication, payload validation, and logging functions to improve robustness and security. The Cloud Layer coordinates aggregation control, fairness regulation, and global model management.

Experimental evaluation demonstrated that the proposed framework provides a practical trade-off between detection effectiveness and training efficiency under heterogeneous IoT set-

tings. Compared with conventional synchronous federated learning, KAC-FL significantly reduced overall training time while maintaining competitive intrusion detection performance. Compared with fully asynchronous aggregation, the proposed framework improved aggregation stability and gateway participation fairness. These results indicate that KAC-FL is a feasible and effective framework for privacy-preserving intrusion detection in distributed IoT environments.

6.2 Future Work

To further enhance the practicality, scalability, and robustness of the proposed KAC-FL framework, several research directions can be explored in future work:

- **Adaptive Aggregation Threshold Optimization**

  The current framework relies on a predefined K-min threshold to control asynchronous aggregation timing. Although this design provides a controllable balance between training responsiveness and aggregation stability, static threshold settings may not be optimal under dynamically changing IoT environments. Future work may investigate adaptive threshold optimization mechanisms that automatically adjust aggregation thresholds according to gateway performance, network latency, or workload conditions to further improve training efficiency and convergence stability.

- **Large-Scale Realistic IoT Deployment Validation**

  The current implementation validates the feasibility of KAC-FL under a controlled experimental environment with a limited number of gateways. While the experimental results demonstrate promising performance, real-world IoT deployments may involve significantly larger numbers of heterogeneous devices, more complex traffic distributions, and unstable communication conditions. Future research should evaluate the scalability and robustness of the proposed framework under larger-scale and more realistic deployment scenarios.

- **Staleness-Aware Asynchronous Aggregation**

The current framework reduces outdated contribution issues by retaining only the latest payload from each gateway before aggregation. However, delayed updates may still introduce varying degrees of staleness that are not explicitly considered in the current weighting mechanism. Future work may integrate staleness-aware aggregation strategies to account for delayed payload freshness, thereby improving convergence behavior and aggregation reliability in highly asynchronous environments.

- **Defense Against Malicious or Compromised Participants**

  This study assumes that participating gateways behave honestly during collaborative federated training. In practical distributed environments, compromised or malicious participants may intentionally submit poisoned or manipulated local model contributions to disrupt global model performance. Therefore, incorporating robust defense mechanisms, anomaly detection, or Byzantine-resilient aggregation strategies represents an important future direction for improving the security and trustworthiness of the KAC-FL framework.

TAIPEI TECH

References

[1] S. Ali, Q. Li, and A. Yousafzai, "Blockchain and federated learning-based intrusion detection approaches for edge-enabled industrial iot networks: A survey," *Ad Hoc Networks*, vol. 152, p. 103 320, 2024.

[2] M. Zang, C. Zheng, T. Koziak, N. Zilberman, and L. Dittmann, "Federated learning-based in-network traffic analysis on iot edge," in *2023 IFIP Networking Conference (IFIP Networking)*, 2023, pp. 1–6.

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, 2017, pp. 1273–1282.

[4] J. L. Hernandez-Ramos, G. Karopoulos, E. Chatzoglou, V. Kouliaridis, E. Marmol, A. Gonzalez-Vidal, et al., "Intrusion detection based on federated learning: A systematic review," *ACM Computing Surveys*, vol. 3, pp. 1–65, 2025.

[5] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[6] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, "A survey on federated learning: Challenges and applications," *International Journal of Machine Learning and Cybernetics*, vol. 14, pp. 513–535, 2023.

[7] M. Ge, N. F. Syed, X. Fu, Z. Baig, and A. Robles-Kelly, "Towards a deep learning-driven intrusion detection approach for internet of things," *Computer Networks*, vol. 186, p. 107 784, 2021.

[8] R. Sahay, G Geethakumari, B. Mitra, and N. Goyal, "Investigating packet dropping attacks in rpl-dodag in iot," in *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, 2019, pp. 1–5.

[9] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[10] M. Azalmad, R. El, Ayachi, and M. Biniz, "Unveiling the performance insights: Benchmarking anomaly-based intrusion detection systems using decision tree family algorithms on the cicids2017 dataset," in *International Conference on Business Intelligence*, 2023, pp. 202–219.

[11] C. S. Kalutharage, X. Liu, C. Chrysoulas, and O. Bamgboye, "Utilizing the ensemble learning and xai for performance improvements in iot network attack detection," in *European Symposium on Research in Computer Security*, 2023, pp. 125–139.

TAIPEI TECH

[12] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[13] M. Chen, B. Mao, and T. Ma, "Fedsa: A staleness-aware asynchronous federated learning algorithm with non-iid data," *Future Generation Computer Systems*, vol. 120, pp. 1–12, 2021.

[14] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, et al., "Federated learning with buffered asynchronous aggregation," in *International conference on artificial intelligence and statistics*, 2022, pp. 3581–3607.

[15] T. Zhang, L. Gao, S. Lee, M. Zhang, and S. Avestimehr, "Timelyfl: Heterogeneity-aware asynchronous federated learning with adaptive partial training," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 5064–5073.

[16] J. Liu, J. Jia, T. Che, C. Huo, J. Ren, Y. Zhou, et al., "Fedasmu: Efficient asynchronous federated learning with dynamic staleness-aware model update," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 38, 2024, pp. 13 900–13 908.

[17] Z. Zhou, Y. Li, X. Ren, and S. Yang, "Towards efficient and stable k-asynchronous federated learning with unbounded stale gradients on non-iid data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3291–3305, 2022.

[18] M. Wen, Y. Zhang, P. Zhang, and L. Chen, "Ids-dwkafl: An intrusion detection scheme based on dynamic weighted k-asynchronous federated learning for smart grid," *Journal of Information Security and Applications*, vol. 89, p. 103 993, 2025.

[19] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[20] D. M. J. G., D. Solans, M. Heikkila, A. Vitaletti, N. Kourtellis, A. Anagnostopoulos, et al., *Non-iid data in federated learning: A survey with taxonomy, metrics, methods, frameworks and future directions*, Available: `https://arxiv.org/abs/2411.12377`, 2024.

[21] C. I. for Cybersecurity, (CIC). "Cicids 2017 dataset." Accessed: Jul. 8, 2025. [Online]. Available: `https://www.unb.ca/cic/datasets/ids-2017.html`.

[22] Microsoft. "Windows 11," Accessed: May 28, 2026. [Online]. Available: `https://www.microsoft.com/windows/windows-11`.

[23] M. Roy. "Vmware workstation pro: Now available free for personal use." Accessed: Jul. 5, 2025. [Online]. Available: `https://blogs.vmware.com/cloud-foundation/2024/05/13/vmware-workstation-pro-now-available-free-for-personal-use/`.

TAIPEI TECH

[24] Canonical Ltd. “Ubuntu 22.04 lts (jammy jellyfish) downloads.” Accessed: Jul. 5, 2025. [Online]. Available: https://releases.ubuntu.com/22.04/.

[25] Python Software Foundation. “Python 3.10 documentation,” Accessed: May 28, 2026. [Online]. Available: https://www.python.org/.

[26] A. Ronacher. “Flask documentation,” Accessed: May 28, 2026. [Online]. Available: https://flask.palletsprojects.com/.

[27] L. Richardson and S. Ruby. “Restful web services,” Accessed: May 28, 2026. [Online]. Available: https://www.oreilly.com/library/view/restful-web-services/9780596529260/.

[28] The Pandas Development Team. “Pandas documentation,” Accessed: May 28, 2026. [Online]. Available: https://pandas.pydata.org/.

[29] C. R. Harris, K. J. Millman, S. J. van, der Walt, et al. “Array programming with numpy,” Accessed: May 28, 2026. [Online]. Available: https://numpy.org/.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. “Scikit-learn: Machine learning in python,” Accessed: May 28, 2026. [Online]. Available: https://scikit-learn.org/.

[31] K. Reitz. “Requests: Http for humans,” Accessed: May 28, 2026. [Online]. Available: https://requests.readthedocs.io/.

[32] J. Kirtland. “Blinker documentation,” Accessed: May 28, 2026. [Online]. Available: https://pythonhosted.org/blinker/.

[33] Pallets Projects. “Werkzeug documentation,” Accessed: May 28, 2026. [Online]. Available: https://werkzeug.palletsprojects.com/.

[34] Pallets Projects. “Jinja documentation,” Accessed: May 28, 2026. [Online]. Available: https://jinja.palletsprojects.com/.

[35] openpyxl developers. “Openpyxl documentation,” Accessed: May 28, 2026. [Online]. Available: https://openpyxl.readthedocs.io/.

[36] O. Rainio, J. Teuho, and R. Klén, “Evaluation metrics and statistical tests for machine learning,” *Scientific reports*, vol. 14, no. 1, p. 6086, 2024.

TAIPEI TECH