Hybrid Distributed Learning With Knowledge Distillation for Resource-Efficient Intrusion Detection in Distributed Networks

Cheolhee Park, Kyungmin Park, Jihyeon Song, Soohyung Kim, and Jonghyun Kim

***Abstract*—The rapid evolution of telecommunications has increased network complexity and driven a shift toward decentralized architectures. While this shift introduces new landscapes and opportunities, it also enlarges the attack surface, highlighting the need for adaptive and scalable network security. In this context, artificial intelligence-based network intrusion detection systems (AI-NIDSs) have been extensively investigated to counter the increasing scale and complexity of network threats. Recently, to enable network threat detection in distributed environments, decentralized learning approaches such as federated learning (FL) and split learning (SL) have been actively explored. However, existing approaches impose substantial computational burdens on resource-constrained nodes and manifest inefficiencies in the learning process, which can lead to unstable convergence and noticeable performance degradation. In this article, we propose a novel AI-driven distributed NIDS that considers the computing capabilities of resource-constrained nodes while enabling efficient learning in distributed environments. To address the above challenges, we leverage the split-FL framework and incorporate a knowledge distillation (KD) strategy, with consideration for the objectives of proactive real-time intrusion detection at the network edge. Experiments on a 5G network dataset and an Open RAN dataset demonstrate that the proposed framework can achieve accuracy comparable to a centralized model while reducing local computational overhead and maintaining stable convergence under realistic data distribution scenarios.**

***Index Terms*—Distributed learning, knowledge distillation (KD), network intrusion detection system (NIDS), network security.**

I. Introduction

As MOBILE telecommunications' technology has evolved through successive generations, the architecture and functionality of networks have become increasingly complex and decentralized. This ongoing evolution has accelerated the emergence of distributed network architectures, resulting in massive growth of system-wide traffic and introducing new challenges for intelligent data engineering and resource-aware management. Moreover, as mobile networks transition beyond the 5G paradigm toward the sixth generation, distributed networking is expected to become a key component of next-generation mobile communication systems [1], [2].

Meanwhile, the evolution and structural transformation of network systems have significantly expanded the attack surface accessible to adversaries, which leads to the growing necessity of security mechanisms. Ensuring network security has, therefore, become a critical objective, driving extensive research on network intrusion detection systems (NIDSs). In particular, intensive investigation has been directed toward artificial intelligence (AI)-based approaches, which aim to detect and mitigate previously unseen attacks. However, most existing deep learning (DL) and machine learning (ML)-based NIDS inherently assume centralized environments, limiting their practicality in distributed environments. From this perspective, two primary limitations can arise in a distributed environment: data transmission and real-time threat detection. In a distributed setting, training an AI-NIDS requires transmitting data from each local node to a central node, causing latency and raising privacy concerns. Furthermore, centralized operation of AI-NIDS for threat detection can lead to significant network congestion, degrading real-time detection efficiency and system-level performance.

Recently, to address these constraints, extensive research has focused on designing AI-NIDS tailored for distributed environments. In particular, federated learning (FL) [3] has yielded numerous results in the field of distributed AI-NIDS. Since FL trains models on local nodes without transmitting raw data to a remote server, it enables global model training in a distributed manner while preserving data privacy. In addition, deploying the trained global model to each local node can reduce the threat detection burden previously concentrated on the central server. However, despite these advantages, FL-based approaches often overlook the diverse computational capabilities of individual local nodes. In the FL process, each local node trains the shared global model, which can lead to substantial computational burden when certain nodes operate with inherently limited resources. As a result, computational workload may be inappropriately allocated across local nodes,

Received 15 January 2026; revised 19 April 2026 and 5 June 2026; accepted 10 June 2026. Date of publication 16 June 2026; date of current version 26 August 2026. This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korea government (MSIT) (Research and international collaboration on trust model-based intelligent incident response technologies in 6G open network environment) under Grant RS-2024-00444170. *(Corresponding author: Jonghyun Kim.)*

Cheolhee Park, Kyungmin Park, and Soohyung Kim are with the Cyber Security Research Division, Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea (e-mail: chpark0528@etri.re.kr; kmpark@etri.re.kr; lifewsky@etri.re.kr).

Jihyeon Song and Jonghyun Kim are with the Department of Computer and Information Security, Sejong University, Seoul, South Korea (e-mail: dmon9518@gmail.com; jhk@sejong.ac.kr).

Digital Object Identifier 10.1109/JIOT.2026.3703835

potentially disturbing the training convergence of the global model. To address this challenge, the split learning (SL) approach [4] has been explored as a viable solution (e.g., [5]). However, since SL fundamentally relies on a relay-based training workflow across participants, it can degrade learning efficiency. More notably, these concerns become more critical in distributed network environments where data volume increases exponentially in real time.

In this article, to address the aforementioned limitations, we propose a novel AI-driven distributed NIDS that effectively adapts to the computing capabilities of local nodes while enabling efficient learning in distributed network environments. Specifically, we leverage a hybrid distributed learning framework that integrates the advantages of collaborative learning in FL and the model segmentation approach in SL. In addition to this fundamental distributed learning architecture, we incorporate a knowledge distillation (KD) strategy to optimize the global predictive model, taking into account the resource constraints of local nodes during the inference phase, i.e., real-time detection. To demonstrate the feasibility of our approach, we evaluate the proposed system using two realistic network datasets collected from different sources: the 5G-NIDD dataset [6] and the Open RAN dataset [7]. In the experiments, we analyze the performance of the proposed system under various data distribution scenarios reflecting real-world conditions. The experimental results demonstrate that the proposed system can achieve superior detection performance compared to existing approaches. Furthermore, it attains accuracy comparable to that of a centralized model while maintaining stable convergence under various data distribution scenarios. In addition, the results show that the proposed system not only substantially reduces the computational burden on resource-constrained local nodes during training but also significantly improves operational efficiency during real-time intrusion detection.

The main contributions of this work can be summarized as follows.

1) We design a novel AI-based NIDS by leveraging a hybrid distributed learning framework that integrates FL with SL. This design significantly reduces the computational burden on resource-constrained local nodes while enabling efficient collaborative learning in distributed intrusion detection.
2) From the perspective of real-time intrusion detection, we incorporate a KD strategy into the proposed framework to convert the server-side back-end model into a lightweight version, addressing a real-world challenge that has been largely overlooked in prior studies. By distilling the complex detection model, we show that resource consumption during inference on local nodes can be significantly reduced while maintaining detection performance.
3) We evaluate the proposed system using two up-to-date network datasets—a fully functional 5G mobile network and a wireless network in an Open RAN architecture—under various data distribution scenarios. The experimental results demonstrate its feasibility, efficiency, and applicability in large-scale distributed environments.

The rest of this article is organized as follows. Section II reviews related work, setting the stage for our contributions. Section III presents the necessary background. Section IV describes the proposed methodology in detail, and Section V reports experimental results and evaluations. Section VI discusses the practical implications of the proposed framework, including training efficiency, scalability, and privacy considerations. Finally, Section VII concludes this article and outlines directions for future research.

II. Related Work

In the field of AI-based network intrusion detection, initial studies primarily focused on applying ML and DL algorithms. However, as network infrastructures evolved toward distributed architectures—such as the Internet of Things (IoT), edge computing, and decentralized networks—the limitations of centralized approaches became more evident, including limited scalability, latency, and privacy risks. This paradigm shift has driven growing interest in designing AI-NIDS capable of operating effectively in distributed environments.

Early work on distributed AI-NIDS was mainly based on combining ML/DL classifiers with an FL framework [8], [9], [10], [11]. Building upon these early studies, Zhang et al. [12] proposed an FL-based detection model that incorporates K-Means-based aggregation and SMOTE-ENN resampling to improve communication efficiency and handle class imbalance in DDoS detection. Similarly, Hamdi [13] proposed an IoT-focused FL–CNN IDS that applied SMOTE to mitigate class imbalance and evaluated its performance with multiple aggregation strategies. Subsequent work extended FL-based intrusion detection to IoT-edge settings by focusing on zero-day botnet attacks, assuming that each device observes only a subset of attack types [14]. Another line of research introduced a random forest-based FL framework in which each node trains a local decision tree on network traffic and energy-consumption data, and the central server aggregates these models into a global random forest ensemble [15]. To address computational and communication constraints, Wang et al. [16] combined deep neural networks with FL and mutual information-based feature selection, reducing input dimensionality without degrading accuracy. Another study developed an FL–XGBoost IDS for low-power and lossy networks and analyzed how the relative placement of detection nodes with respect to attackers affects detection rates across attack categories [17]. To prevent reverse engineering of model updates in SDN-based AIoT environments, Ma and Su [18] proposed an autoencoder-based FL framework and applied secure multiparty computation to secure the aggregation process. To mitigate communication overhead and reduce training time in IoT networks, Almarshdi et al. [19] proposed a distributed federated SL-based IDS for DDoS detection, integrating an adaptive aggregation method with a state-based multinode selection scheme.

In the Internet of Medical Things (IoMT) domain, Itani et al. [20] combined traditional ML algorithms with FL to manage device heterogeneity and preserve data locality in

healthcare networks. To fine-tune a global detection model on local healthcare data, several studies adopted federated transfer learning to personalize the shared model at each node and reduce the overall training cost [21], [22]. In a subsequent effort, Fahim-Ul-Islam et al. [23] proposed an FL framework that integrates metalearning with a robust aggregation strategy and employs a Kolmogorov–Arnold-based convolutional network to improve interpretability and enhance feature representation on resource-constrained IoMT devices. Focusing on Industrial IoT (IIoT) environments, Khoa et al. [24] proposed a collaborative intrusion detection framework that integrates distributed learning with deep belief networks, enabling subnetwork-specific training through smart filters deployed at IoT gateways. Li et al. [25] proposed a distributed intrusion detection model for IIoT environments that combines FL with a gated recurrent unit (GRU) architecture and adopts an attacker-centric mitigation strategy by distributing defense intelligence along attack paths. To enhance privacy in IIoT environments, Ruzafa-Alcázar et al. [26] incorporated differential privacy (DP) mechanisms into a federated intrusion detection system. In a related direction, another line of privacy-preserving research introduced secure learning frameworks based on the Paillier cryptosystem [27], [28].

Focusing on next-generation network environments, recent studies introduced hierarchical and hybrid FL-based models for Beyond-5G networks, employing lightweight domain-level detectors and hybrid DL architectures to handle heterogeneous traffic [29], [30]. To address non-IID and imbalanced data across local and global levels, Singh et al. [31] proposed an enhanced FL-based intrusion detection system that integrates balanced sampling, feature augmentation, and KD. Departing from prior FL-based approaches, Park et al. [5] proposed an SL-based intrusion detection framework for next-generation mobile networks. To accommodate resource-constrained local environments, they adopted a model-split approach, where the front-end (shallow) model is deployed on local nodes, and the back-end model is processed by a central server. In Open RAN scenarios, Attanayaka et al. [32] proposed a peer-to-peer FL (P2P FL) framework for anomaly detection, aiming to enable decentralized and privacy-preserving management. The framework incorporated multiple P2P FL variants tailored for RAN Intelligent Controllers to reflect the modular and distributed architecture of the network. To support sustainable detection of DDoS attacks in Open RAN environments, Benzaïd et al. [33] proposed a federated continual learning framework that incorporates a rehearsal-based continual learning strategy. A related effort introduced a secure P2P FL-based intrusion detection system for Open RAN that incorporates Secure Average Computation to protect the aggregation process [34]. To support efficient learning, the system applied a peer selection mechanism that allowed only high-performing agents to participate in each aggregation round and distributed the updated global model to the remaining agents through transfer learning. Focusing on 6G IoT-edge networks, Pithani and Rout [35] proposed a clustered FL-based intrusion detection system that forms virtual clusters of edge nodes based on model similarity and applies an adaptive training strategy that adjusts local epochs according to data imbalance.

While previous efforts have advanced intrusion detection across various domains, most studies do not jointly address in-field computational constraints at local nodes and training efficiency under distributed settings. Furthermore, operational efficiency during real-time inference has generally been neglected, despite its critical importance under resource-constrained conditions. To address these challenges, we propose a novel distributed AI-based intrusion detection framework that enhances both training and inference efficiency.

III. Background

In this section, we briefly illustrate the core concepts of distributed learning and KD that serve as the basis of the proposed framework.

A. Distributed Learning Framework

As distributed computing expands, local data generation has surged, presenting both opportunities and challenges in scalability, communication, and privacy. These constraints have exposed the limitations of centralized learning and accelerated the shift toward distributed paradigms.

FL [3] has emerged as a practical solution for enabling collaborative model training across decentralized data sources without sharing raw data. Since FL transmits only model parameters—such as weights or gradients—instead of raw data, it enables privacy-preserving training across multiple participants in distributed environments. In the FL framework, a central server and multiple clients collaboratively participate in training a shared global model. The central server initializes the global model and distributes it to multiple clients. Each client trains the model locally using its own data, resulting in an updated model. After completing local training, clients send only their updated model parameters back to the server. The server then aggregates these local models, typically via federated averaging (FedAvg). This training cycle is repeated over multiple rounds, progressively improving the global model toward better performance and generalization. While FedAvg remains the most widely adopted aggregation method, alternative methods have been proposed to improve training convergence, robustness, and performance (e.g., FedProx [36] and Multi-Krum [37]).

Although FL enables collaborative training without sharing raw data in distributed environments, handling the entire model on each local node often incurs significant computational overhead, particularly in resource-constrained environments. SL [4] mitigates this issue by dividing the model into a front-end segment $f(\cdot;\theta_f)$ on the client and a back-end segment $g(\cdot;\theta_g)$ on the server. In the SL framework, each client processes its local data x_k through the front-end model to produce intermediate activations $z_k = f(x_k;\theta_f)$, which are transmitted to the server for forward and backward computation. The server then computes the gradients $\partial\mathcal{L}_k/\partial z_k$ with respect to the cut layer and sends them back to the client. Upon receiving these gradients, the client updates its front-end parameters by computing $\nabla_{\theta_f}\mathcal{L}_k = \partial\mathcal{L}_k/\partial z_k \cdot \partial z_k/\partial\theta_f$. This split architecture reduces local computation while maintaining data locality.

While FL and SL support distributed training without sharing raw data, FL requires each node to train the full model locally, whereas SL follows a sequential training process. As a result, FL imposes high computational costs on resource-limited nodes, while SL suffers from inefficiency due to its relay-based nature. To address these issues, the split-fed learning (SFL) framework [38] combines the model-splitting strategy of SL with the aggregation mechanism of FL. In SFL, each node trains a front-end model and sends intermediate activations (smashed data) to the server. The server then continues the forward and backward passes on its back-end model, returning gradients for local updates. Unlike the sequential process of SL, SFL enables parallel local updates through a federation step, where the server aggregates the front-end parameters (e.g., via FedAvg) and redistributes the global model to all nodes.[1] Formally, the front-end aggregation at round t can be expressed as follows:

$$\theta_f^{(t+1)} = \sum_{k=1}^{K} \frac{n_k}{n} \theta_{f,k}^{(t)} \tag{1}$$

where $\theta_{f,k}^{(t)}$ denotes the front-end model of client k at round t, n_k denotes the number of local samples at client k, and n is the total number of samples across all K participating clients. This hybrid learning process allows the SFL framework to benefit from the computational efficiency of SL on local nodes and the collaborative generalization capability of FL across multiple data sources. Building on this foundation, several variants of the SFL framework—such as FSL-SAGE [39] and HSFL [40]—have been proposed to enhance model personalization, scalability, or privacy in diverse settings.

B. Knowledge Distillation

The increasing architectural complexity and predictive capacity of state-of-the-art DL models have raised significant challenges in terms of computational efficiency and model optimization. To address these challenges, various approaches have been explored to reduce model complexity while preserving performance, such as model pruning, quantization, and transfer learning. In this regard, KD [41] is widely recognized as a practical approach for transferring knowledge from a high-capacity model to a compact and efficient one, enabling significant model compression without substantial loss in performance. The core concept of KD is to train a smaller student model to emulate the output behavior of a larger teacher model, with the goal of transferring the generalization capability of the teacher into a more compact form. In the KD framework, training involves two models: a pretrained *teacher model* θ_t and a trainable *student model* θ_s. Rather than relying solely on ground-truth labels (i.e., hard targets), the student learns from soft targets generated by the teacher—typically, class probability distributions obtained via temperature-scaled softmax. Let $\mathbf{v} = f(x;\theta_t) \in \mathbb{R}^c$ be the output logits of the teacher model for input x, where c is the number of classes. The corresponding softened output vector is then given by

$$\tilde{\mathbf{p}}_t(x) = \left\{\tilde{p}_{t,i}(x)\right\}_{i=1}^{c} = \left\{\frac{\exp(v_i/T)}{\sum_{j=1}^{c}\exp\left(v_j/T\right)}\right\}_{i=1}^{c} \tag{2}$$

where $T > 0$ is a temperature parameter that controls the smoothness of the output distribution. Higher temperatures yield softer probability vectors that reveal interclass similarities and, thus, provide informative learning signals to the student model. The student is trained to match both the hard targets and the soft targets. To this end, the loss function is formulated as a weighted sum of the standard cross-entropy loss, which measures the distance between model predictions and ground-truth labels, and a distillation loss based on the Kullback–Leibler divergence between the teacher and student output distributions

$$\begin{aligned}\mathcal{L}_{\mathrm{KD}} = (1-\lambda)\cdot\mathcal{L}_{\mathrm{CE}}(\mathbf{p}_s(x), y) \\ + \lambda T^2\cdot\mathcal{L}_{\mathrm{KL}}(\tilde{\mathbf{p}}_t(x)\,\|\,\tilde{\mathbf{p}}_s(x))\end{aligned} \tag{3}$$

where $\lambda \in [0,1]$ is a tunable hyperparameter that balances the contribution of each loss term, and $\tilde{\mathbf{p}}_s(x)$ and $\mathbf{p}_s(x)$ denote the softened and standard output distributions of the student model, respectively.[2]

This joint supervision allows the student to capture both the ground-truth information and the interclass relational knowledge embedded within the teacher model. As a result, KD not only improves the generalization ability of the student model but also enables effective model compression without substantial degradation in performance.

IV. Proposed Methodology

Fig. 1 illustrates the overall framework of the proposed distributed AI-based NIDS, which consists of three main processes: local data processing, resource-aware training, and model optimization.

A. Local Data Processing

In the proposed framework, we assume that each local node is responsible for its own data processing pipeline, which is carried out independently prior to participating in collaborative model training. This local preprocessing step performs outlier removal and feature scaling to convert raw attributes into model-compatible formats.

To improve distributional consistency and enhance training stability, the system first removes statistical outliers from numerical attributes based on the median absolute deviation (MAD). As a robust measure of statistical dispersion, MAD is less sensitive to extreme values than standard deviation and provides a reliable basis for detecting anomalous data points in skewed or heavy-tailed distributions. For a given numeric attribute $\mathcal{A} = \{x_1, x_2, \ldots, x_n\}$, the MAD of $\mathcal{A}$ is defined as follows:

$$\mathrm{MAD}_{\mathcal{A}} = \mathrm{median}\left(|x_i - \mathrm{median}(\mathcal{A})|\right). \tag{4}$$

[1] In the SFL framework [38], the system distinguishes between a main server, which trains the back-end model, and a separate Fed server, which performs client-side aggregation. As an alternative design, the framework also allows these roles to be unified in a single server.

[2] That is, $\tilde{\mathbf{p}}_s(x) = \mathrm{softmax}((g(x;\theta_s))/T)$, and $\mathbf{p}_s(x)$ denotes the standard output with $T = 1$.

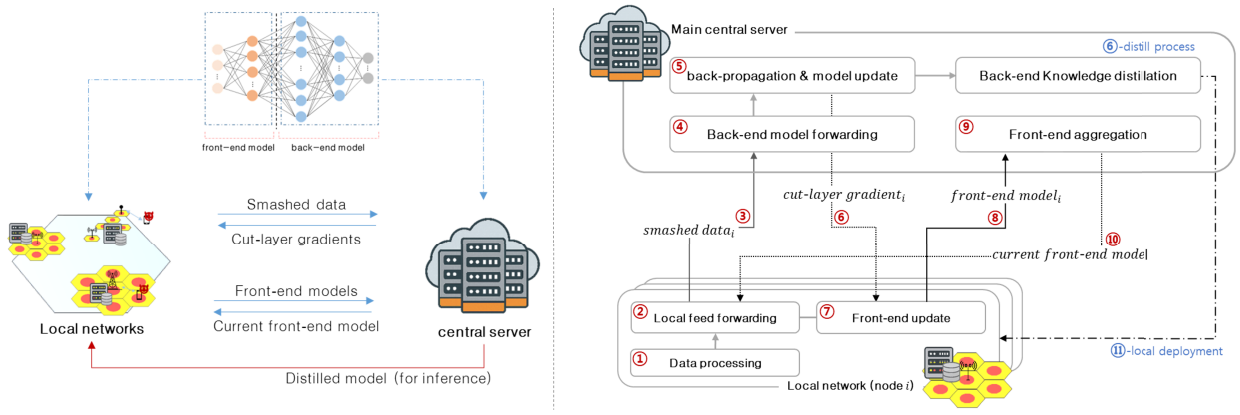

Fig. 1. Overall architecture and workflow of the proposed SFL-KD framework (left: conceptual overview; right: step-by-step workflow).

Assuming a normal distribution, the standard deviation can be estimated as $\hat{\sigma} = 1.4826 \times \text{MAD}$. We regard a value as an outlier if its absolute deviation from the median exceeds $10 \times \sigma$ and remove such instances accordingly.[3] Note that the process of outlier removal is conducted independently for each class and should be performed prior to the feature scaling step (i.e., normalization or standardization). Following outlier removal, the system encodes categorical attributes using one-hot encoding. In this scheme, each categorical value is represented as a binary vector with a single active element corresponding to the observed category.

Following these steps, the system scales all numerical attributes to promote stable convergence during model training. In general, several standard scaling techniques can be considered, such as min–max normalization and standardization. However, in a distributed setting, data distributions may vary significantly across local nodes, which can undermine the effectiveness of scaling methods that rely on population statistics. That is, when normalization or standardization is performed using locally computed statistics, it may lead to inconsistencies in feature importance across different nodes and semantic mismatch in feature representation. To address this challenge, we adopt a consistent scaling method that avoids reliance on global population statistics and remains stable under variations in local data distributions. Specifically, we apply robust scaling to standardize numerical attributes in a manner that is less sensitive to outliers and distributional shifts. In robust scaling, each numerical attribute is scaled based on its median and interquartile range (IQR). The corresponding transformation for each attribute $\mathcal{A} = \{x_1, x_2, \ldots, x_n\}$ is given by

$$\tilde{x}_i = \frac{x_i - \text{median}(\mathcal{A})}{\text{IQR}(\mathcal{A})}. \tag{5}$$

Since robust scaling is grounded in distribution-invariant statistics, it enables consistent scaling across heterogeneous nodes. After completing the above preprocessing steps, each node prepares its local data for collaborative training.

[3]The threshold multiplier of 10 is chosen to conservatively filter extreme outliers while minimizing the risk of removing data points that are statistically valid but lie on the boundary of the global distribution.

Algorithm 1 SFL-KD for Intrusion Detection: Overall Training Procedure

Input: Local datasets $\{\mathcal{D}_k\}_{k=1}^{K}$, time index $t = 0, \ldots, R$

if $t = 0$ **then**
  Server operations:
  Initialize global model $\mathbf{W}^{(0)} = (\mathbf{W}_f^{(0)}, \mathbf{W}_b^{(0)})$
  Initialize student model $\boldsymbol{\theta}^{(0)}$ corresponding to $\mathbf{W}_b^{(0)}$
  Distribute front-end model $\mathbf{W}_f^{(0)}$ to all local nodes
else
  Joint operations:
  for each local node k in parallel **do**
    Run **JointTrain** with $\mathcal{D}_k, \mathbf{W}^{(t)}, \boldsymbol{\theta}^{(t)}$(Alg. 2)
    Send updated $\mathbf{W}_{f,k}^{(t)}$ with n_k to the server
  end for
  Server operations:
  Aggregate front-end model: $\mathbf{W}_f^{(t+1)} = \sum_{k=1}^{K} \frac{n_k}{n} \cdot \mathbf{W}_{f,k}^{(t)}$
  Distribute updated $\mathbf{W}_f^{(t+1)}$ to all local nodes
end if

Output: $\boldsymbol{\theta}^{(R)}$, $\mathbf{W}^{(R)} = (\mathbf{W}_f^{(R)}, \mathbf{W}_b^{(R)})$

B. Resource-Aware Training for Distributed NIDS

Despite the growing adoption of distributed learning paradigms in the field of network intrusion detection, several unresolved challenges remain in practice. Most notably, the computational and memory requirements of DL models often exceed the capabilities of edge devices—particularly when running alongside other local processes—making full-model training infeasible on resource-constrained nodes. While split architectures can reduce the computational load on local nodes, the relay-based nature of SL frameworks may lead to inefficient training and introduce delays in global model updates. Departing from conventional approaches, we adopt a hybrid distributed learning framework as the foundation of our system, leveraging the strengths of FL and SL to effectively mitigate the computational load on local nodes while supporting an efficient training process across the entire network.

Algorithm 1 describes the global coordination procedure for the proposed SFL-KD framework in intrusion detection. To

Algorithm 2 **JointTrain**: Model Update Procedure

Input: Local dataset $\mathcal{D}_k = \{\mathbf{x}_k, \mathbf{y}_k\}$, front-end model $\mathbf{W}_f^{(t)}$, back-end model $\mathbf{W}_b^{(t)}$, student model $\boldsymbol{\theta}^{(t)}$

Local node operations:

Compute smashed data $\mathbf{z}_k = f(\mathbf{x}_k; \mathbf{W}_f^{(t)})$

Send $\{\mathbf{z}_k, \mathbf{y}_k\}$ to the server

Server operations:

Compute prediction $\hat{\mathbf{y}} = g(\mathbf{z}_k; \mathbf{W}_b^{(t)})$

Compute gradient $\nabla_{\mathbf{W}_b^{(t)}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}_k) = \frac{\partial \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}_k)}{\partial \mathbf{W}_b^{(t)}}$

Update $\mathbf{W}_b^{(t+1)} \leftarrow \mathbf{W}_b^{(t)} - \eta_b \cdot \nabla_{\mathbf{W}_b^{(t)}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}_k)$

Run **Distillation** with $\{\mathbf{z}_k, \mathbf{y}_k\}, \mathbf{W}_b^{(t+1)}, \boldsymbol{\theta}^{(t)}$(Alg. 3)

Send $\frac{\partial \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}_k)}{\partial \mathbf{z}_k}$ to the client

Local node continues:

Compute local gradient: $\nabla_{\mathbf{W}_f^{(t)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_k} \cdot \frac{\partial \mathbf{z}_k}{\partial \mathbf{W}_f^{(t)}}$

Update $\mathbf{W}_f^{(t)} \leftarrow \mathbf{W}_f^{(t)} - \eta_f \cdot \nabla_{\mathbf{W}_f^{(t)}}$

Output: Updated $\mathbf{W}_{f,k}^{(t)}$, $\mathbf{W}_b^{(t+1)}$, and $\boldsymbol{\theta}^{(t+1)}$

enable collaborative training while minimizing local computational load, we adopt a hybrid distributed learning strategy integrated with KD. The algorithm begins with the server initializing the global intrusion detection model $\mathbf{W}$ and partitioning it into a front-end model $\mathbf{W}_f$, which is assigned to all local nodes, and a back-end model $\mathbf{W}_b$, which remains on the server. Concurrently, the server initializes a lightweight student model $\boldsymbol{\theta}$ that is trained to approximate the behavior of the server-side back-end model. Once initialization is complete, the front-end model $\mathbf{W}_f$ is distributed to all local nodes. In the subsequent training round $t > 0$, each local node k collaborates with the server to execute the **JointTrain** process (see Algorithm 2) using its local dataset $\mathcal{D}_k$. During this process, each local node performs a forward pass through its front-end model and transmits the resulting intermediate activations $\mathbf{z}_k = f(\mathbf{x}_k; \mathbf{W}_f^{(t)})$ along with the corresponding labels $\mathbf{y}_k$ to the server. The server then computes the gradient at the cut layer via back-propagation through its back-end model and returns the resulting gradient to the corresponding local node. Upon receiving the gradient from the server, the local node proceeds to update its front-end model $\mathbf{W}_f^{(t)}$ by continuing back-propagation through the front-end layers. This joint procedure between the local node and the server is repeated for a predefined number of local iterations. After completing the update of the front-end model, each local node transmits its updated front-end parameters $\mathbf{W}_{f,k}^{(t)}$ along with the corresponding sample size n_k to the server for global refinement of the shared model. Note that this sequence of local node-server interactions is executed in parallel across all participating local nodes. Upon receiving the updated front-end models $\{\mathbf{W}_{f,k}^{(t)}\}_{k=1}^{K}$ from each local node, the server aggregates them via a weighted average using the standard FedAvg rule. The refined global front-end model $\mathbf{W}_f^{(t+1)}$ is then redistributed to all local nodes. This global training process is repeated for R rounds, resulting in the final trained global model $\mathbf{W}^{(R)} = (\mathbf{W}_f^{(R)}, \mathbf{W}_b^{(R)})$ and the distilled back-end model $\boldsymbol{\theta}^{(R)}$ for distributed intrusion detection.

Considering detection efficiency during real-time inference, we incorporate KD in the training procedure to compress the server-side back-end model into a lightweight student model. Specifically, we adopt an online KD strategy, where the student model is updated simultaneously with the training of the server-side back-end model. This approach avoids the need for a post hoc distillation process and enables the dynamic transfer of knowledge in a naturally integrated manner during joint training. Indeed, in conventional offline distillation, the intermediate data required for supervision—namely, the smashed data and corresponding hard labels—must be retained throughout the entire training phase, imposing additional memory and management overhead on the system. Moreover, even after global model convergence, extra workload can be introduced in selecting appropriate data samples or representative local nodes to construct a teacher–student training set. To circumvent these burdens and simplify the overall pipeline, we adopt an online distillation strategy that naturally integrates student learning into the primary training process. Algorithm 2 illustrates the joint training procedure between each local node and the central server, where KD is incorporated as part of the server-side process. This procedure is invoked during each round of Algorithm 1 via the **JointTrain** call. For each minibatch $\mathcal{D}_k = \{\mathbf{x}_k, \mathbf{y}_k\}$, the local node begins by computing the intermediate activation $\mathbf{z}_k = f(\mathbf{x}_k; \mathbf{W}_f^{(t)})$ through its front-end model $\mathbf{W}_f^{(t)}$. The resulting smashed data $\mathbf{z}_k$, along with its corresponding label $\mathbf{y}_k$, is then transmitted to the server. Upon reception, the server conducts a forward pass through its back-end model $\mathbf{W}_b^{(t)}$ to compute the prediction $\hat{\mathbf{y}} = g(\mathbf{z}_k; \mathbf{W}_b^{(t)})$ and subsequently calculates the gradient $\nabla_{\mathbf{W}_b^{(t)}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}_k)$. This gradient is used to update the back-end model via gradient descent.

After the back-end model update, the server invokes the **Distillation** routine (see Algorithm 3), which is augmented with an additional shadowing step beyond standard online distillation. The distillation process consists of two parts: online distillation and back-end shadowing. In the online distillation step, the server computes the temperature-scaled soft target $\tilde{\mathbf{p}}_t$ from the updated back-end model $\mathbf{W}_b^{(t+1)}$ and the softened output $\tilde{\mathbf{p}}_s$ from the student model $\boldsymbol{\theta}^{(t)}$.[4] The student is then optimized with the distillation loss $\mathcal{L}_{\text{KD}}$, defined as a weighted combination of the cross-entropy loss with the ground-truth label and the Kullback–Leibler divergence between $\tilde{\mathbf{p}}_t$ and $\tilde{\mathbf{p}}_s$. In the shadowing step, the server samples synthetic activations $\mathbf{z}_m \sim \mathcal{Q}$ from a prior distribution approximating the activation space. These synthetic activations are processed by both the back end and the student to generate confidence vectors $\mathbf{p}_t$ and $\mathbf{p}_s$. The student parameters are then updated with the cross-entropy loss $\mathcal{L}_{CE}(\mathbf{p}_t, \mathbf{p}_s)$, which enforces prediction consistency between the student and the back end beyond the observed data. In its original context, model shadowing was introduced as an adversarial technique for replicating the behavior of a victim model. In this work, we reinterpret

[4] $\tilde{g}(\cdot)$ and $\tilde{h}(\cdot)$ denote the logit outputs of the back-end and student models, respectively, before the softmax operation. The temperature-scaled predictions $\tilde{\mathbf{p}}_t$ and $\tilde{\mathbf{p}}_s$ are obtained by applying softmax to these logits.

Algorithm 3 Distillation: Shadow-Augmented KD (Server-Side)

Input: Smashed data $\{\mathbf{z}_k, \mathbf{y}_k\}$, back-end $\mathbf{W}_b^{(t+1)}$, student $\boldsymbol{\theta}^{(t)}$

Back-end knowledge distillation:

Compute soft target $\tilde{\mathbf{p}}_t = \text{softmax}(\tilde{g}(\mathbf{z}_k; \mathbf{W}_b^{(t+1)})/T)$

Compute soft output $\tilde{\mathbf{p}}_s = \text{softmax}(\tilde{h}(\mathbf{z}_k; \boldsymbol{\theta}^{(t)})/T)$

Update $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta_1 \cdot \nabla_{\boldsymbol{\theta}^{(t)}} \mathcal{L}_{KD}(\mathbf{p}_s, \mathbf{y}_k, \tilde{\mathbf{p}}_t, \tilde{\mathbf{p}}_s)$

Shadow-augmented step:

Sample synthetic activations $\{\mathbf{z}_m\}_{m=1}^{M} \sim \mathcal{Q}$

Compute synthetic confidence vectors: $\hat{\mathbf{p}}_t = g(\mathbf{z}_m; \mathbf{W}_b^{(t+1)})$ and $\hat{\mathbf{p}}_s = h(\mathbf{z}_m; \boldsymbol{\theta}^{(t)})$

Update $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta_2 \cdot \nabla_{\boldsymbol{\theta}^{(t)}} \mathcal{L}_{CE}(\hat{\mathbf{p}}_t, \hat{\mathbf{p}}_s)$

Output: Updated student $\boldsymbol{\theta}^{(t+1)}$

and repurpose shadowing as a regularization mechanism, introducing the shadow-augmented step to strengthen student–teacher alignment beyond the observed activations. Conceptually, this step shares similarities with data-free and zero-shot KD, as it enforces prediction consistency without relying solely on labeled real data. However, unlike those offline approaches, our design integrates shadowing into the online distillation process, yielding a lightweight yet effective extension that mitigates overfitting and enhances robustness.

Once the back-end model update and distillation step are complete, the server returns the gradient $(\partial \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}_k))/(\partial \mathbf{z}_k)$, computed at the cut layer, to the corresponding local node. This bidirectional interaction can be iterated for a predefined number of local update steps,[5] yielding updated parameters for the front-end model $\mathbf{W}_{f,k}^{(t)}$, the server-side back-end model $\mathbf{W}_b^{(t+1)}$, and the student model $\boldsymbol{\theta}^{(t+1)}$. Note that the gradient at the cut layer, computed during the back-propagation of the back-end model, can be returned to the local node immediately after the update of $\mathbf{W}_b^{(t)}$, independent of the subsequent distillation process. In other words, the local-side back-propagation can proceed in parallel with the server-side distillation step.

Through the training process, each component of the distributed NIDS—including the front-end model, the server-side back-end model, and the distilled lightweight model—is iteratively optimized to detect network intrusion across distributed environments. Finally, the trained lightweight student model is deployed to each local node, where the corresponding predictive model enables efficient real-time intrusion detection without reliance on the centralized server.

C. Real-Time Intrusion Detection at the Inference Stage

Following the completion of the collaborative training process, the trained lightweight model $\boldsymbol{\theta}^{(R)}$ is deployed to each local node. To enable efficient detection directly within the local network, the student model is designed to operate independently of the centralized server and to function in conjunction with the front-end model residing on each local node. Concretely, given an incoming network traffic instance x, the local node first processes the input through its front-end model $\mathbf{W}_f^{(R)}$ to generate an intermediate activation $z = f(x; \mathbf{W}_f^{(R)})$. These smashed data z are then passed to the deployed student model $\boldsymbol{\theta}^{(R)}$, which produces the final prediction on whether the incoming input corresponds to an attack or benign behavior. Although the inference process may seem structurally separated between the front-end and student models, they jointly constitute a unified predictive function, with a seamless transition between their respective input and output spaces.

This structural integration is further justified by the fact that the student model $\boldsymbol{\theta}^{(R)}$ is explicitly trained to approximate the output behavior of the original back-end model $\mathbf{W}_b^{(R)}$. Given that both models share a common input $z = f(x; \mathbf{W}_f^{(R)})$, the student model learns to replicate the functional mapping $g(z; \mathbf{W}_b^{(R)})$ through distillation-based supervision during training. As a result, the composed function $h(f(x; \mathbf{W}_f^{(R)}); \boldsymbol{\theta}^{(R)})$ asymptotically approximates the output $\tilde{y} = g(f(x; \mathbf{W}_f^{(R)}); \mathbf{W}_b^{(R)})$ of the original global model. More formally, let f, g, and h be the front-end, back-end, and student models, respectively, with corresponding parameters $\mathbf{W}_f$, $\mathbf{W}_b$, and $\boldsymbol{\theta}$. Suppose that $z = f(x; \mathbf{W}_f)$ is the intermediate representation of input x, and assume that $h(z; \boldsymbol{\theta})$ is trained via KD using soft targets generated by $g(z; \mathbf{W}_b)$. Then, for any input x drawn from the data distribution $\mathcal{D}$, the output of the student inference $y = h(f(x; \mathbf{W}_f); \boldsymbol{\theta})$ approximates the output of the global model $\tilde{y} = g(f(x; \mathbf{W}_f); \mathbf{W}_b)$, i.e.,

$$h\left(f\left(x; \mathbf{W}_f\right); \boldsymbol{\theta}\right) \approx g\left(f\left(x; \mathbf{W}_f\right); \mathbf{W}_b\right).$$

This functional alignment ensures that the final prediction at the local node remains consistent with that of the global model, which, in turn, demonstrates the reliability of the lightweight student model as an approximation of the prediction produced by the original global model. Notably, since inference operates entirely at the local node without server communication, the real-time detection capability is structurally independent of network transport dynamics.

From a system-wide perspective, the proposed framework offers a resource-efficient training and inference pipeline for intrusion detection across distributed environments. By integrating hybrid distributed learning with KD, the system enables collaborative model optimization while taking into account the computational constraints of local nodes. Unlike prior approaches that leverage transfer learning in a post hoc manner, the proposed system embeds the distillation process within the training loop to achieve seamless knowledge transfer from the global detection model.

V. Experiments and Evaluations

To evaluate the effectiveness of the proposed framework, we conduct a series of comprehensive experiments under various data distribution scenarios.

A. Dataset Description

To evaluate the performance, we adopt two benchmark datasets that represent distinct distributed network environments reflecting real-world operating conditions.

[5]The distillation process can be repeated for a given number of local-independent iterations, which can also be adjusted adaptively. In our experiments, we fixed the number of subiterations to 5.

TABLE I
DATA DISTRIBUTION OF THE 5G-NIDD DATASET

Type	Class	Rows	Weight (%)
Normal	Benign	477,737	39.29%
Attack	HTTP Flood	140,812	11.58%
	ICMP Flood	1,155	0.09%
	SYN Flood	9,721	0.79%
	UDP Flood	457,340	37.61%
	Slowrate DoS	73,124	6.01%
	SYN Scan	20,043	1.64%
	TCP Connect Scan	20,052	1.64%
	UDP Scan	15,906	1.30%
Total		1,215,890	100.00%

TABLE II
DATA DISTRIBUTION OF THE OPEN RAN DATASET

Type	Class	Rows	Weight (%)
Normal	Web Browsing	238,140	7.50%
	VoIP	383,769	12.09%
	IoT Messaging	508,187	16.01%
	YouTube Streaming	319,119	10.05%
Attack	DDoS-Ripper	490,425	15.45%
	DoS-Hulk	564,008	17.76%
	PortScan	461,151	14.52%
	Slowloris	210,341	6.62%
Total		3,175,140	100.00%

1) 5G-NIDD: The 5G-NIDD dataset was obtained from a fully operational 5G test network and offers realistic network traffic data [6]. The dataset includes a total of 1 215 890 network flows, comprising realistic benign traffic and eight distinct attack scenarios. The attack classes are broadly categorized into denial-of-service (DoS) and port scanning. The DoS category includes ICMP flood, UDP flood, SYN flood, HTTP flood, and slow-rate attacks, while the port scanning class consists of SYN scan, TCP connect scan, and UDP scan. Each data point consists of 25 raw features that reflect key behaviors at the network and transport layers. Note that we excluded irrelevant fields that do not meaningfully affect model performance (e.g., row indices). As described in Section IV-A, the dataset is preprocessed to convert the raw data into a model-interpretable representation. Specifically, the refined dataset contains a total of 91 features derived from both categorical and numerical fields. Table I summarizes the classwise distribution of the dataset.

2) Open RAN Dataset: The Open RAN dataset was constructed from a fully operational wireless testbed built on the OpenIreland platform, emulating real-world Open RAN networks [7]. The testbed comprises a virtualized radio access architecture implemented with srsRAN and SDR interfaces and supports the measurement of real-time RAN KPIs under controlled traffic and mobility conditions. During data collection, the UEs were configured to operate under various mobility patterns—including static, pedestrian, vehicular, and high-speed scenarios—while generating both benign and adversarial network traffic. The dataset includes a total of 3 175 140 samples, each representing a snapshot of RAN behavior at the base station. These samples encompass a wide range of physical-layer and MAC-layer KPIs. Specifically, there are eight traffic types in the dataset, covering four benign applications and four attack types. Similar to the 5G data, the Open RAN dataset is preprocessed to convert the collected measurements into a model-interpretable format. Table II summarizes the number of samples per class label, providing an overview of the classwise data distribution.

In the experiments, each dataset is partitioned into training and test sets with an 8:2 ratio. The training set is further divided into multiple subsets to simulate different data distribution scenarios.

B. Experimental Setup

To evaluate the robustness of the proposed framework, we explore three data distribution scenarios that reflect real-world conditions: independent and identically distributed (IID), non-identically distributed (non-IID), and extremely skewed. In the IID setting, each node is assigned a uniformly sampled subset of the full dataset, preserving the global class distribution. By contrast, the non-IID scenario represents a setting where each node is assigned a subset of data biased toward a limited number of classes, resulting in a skewed local distribution that deviates from the global class distribution.[6] In the extremely skewed setting, we assume that only single-class events occur at each local node, leading to strong local bias and extreme heterogeneity. Based on these scenarios, we focus on the IID and non-IID settings for the Open RAN dataset to analyze the impact of heterogeneous data distributions across local nodes. For the 5G-NIDD dataset, we consider the IID and extremely skewed settings to examine the performance of the proposed framework under severe class imbalance, which is more representative of large-scale and imbalanced intrusion detection environments.

For the distributed setting, we configured the number of local nodes to match the number of semantic classes. To build distributed environments, we employed container-based virtualization, where each local node operated as an isolated instance with dedicated data and training processes. Specifically, we implemented eight local nodes for experiments on the Open RAN dataset, corresponding to its eight classes. For the 5G-NIDD dataset, we constructed eight nodes based on detailed attack types and uniformly distributed benign traffic across all nodes to better reflect real-world conditions. All experiments were repeated ten times under the same configuration, and the reported results correspond to the best-performing model observed across repeated runs. The experiments were conducted on a server equipped with an Intel Xeon Silver 4214 CPU (48 cores, 2.20 GHz) and NVIDIA GeForce RTX 3090 GPUs. The system was operated on Ubuntu 22.04.5 LTS, and all models were implemented in Python using the PyTorch framework with GPU acceleration enabled.

[6]We simulate non-IID data partitions using a Dirichlet distribution. To induce a strong class imbalance across local nodes, the concentration parameter α is fixed to 0.3 in our experiments.

TABLE III
ARCHITECTURES OF THE GLOBAL AND LIGHTWEIGHT MODELS

Model	Architecture
Global Full Model	FC(256) → FC(128) → Conv1D(128, k=5) → Conv1D(64, k=3) → Conv1D(32, k=2) → FC(256) → FC(128) → FC(64) → FC(C)
Lightweight Model	Conv1D(64, k=3) → FC(128) → FC(64) → FC(C)

To account for realistic network dynamics, we further assumed random node dropouts during training, where 20% of the local nodes became unavailable in each round. This setting reflects practical scenarios in which local nodes may become temporarily unreachable due to connectivity disruptions, resource unavailability, or maintenance operations, which are common in real-world distributed network environments. Under this condition, the remaining active nodes continue to participate in collaborative training, while the absent nodes rejoin in subsequent rounds, resulting in a dynamic and inconsistent set of participants across training rounds.

C. Intrusion Detection Model Configuration

The proposed framework adopts a modular detection architecture composed of a feature projection block, a convolutional feature extractor, and a fully connected classification module. The projection block consists of two fully connected layers that transform preprocessed network traffic features into a low-dimensional latent representation, enabling effective feature interaction and dimensional alignment. The convolutional block consists of three 1-D convolutional layers with max-pooling operations, capturing local structural dependencies within the latent representation. The classification module, composed of three fully connected layers, determines whether the observed traffic corresponds to benign or malicious behavior. Based on this full model configuration, local nodes and the central server collaboratively train the global detection model in a distributed manner.

For resource-efficient real-time operation at local nodes, the framework incorporates an online distillation process in which the knowledge of the trained global detection model is distilled into a lightweight model. In our experiments, the lightweight model is configured with a single 1-D convolutional layer followed by two fully connected layers. This design strikes a balance between representational capacity and computational efficiency, allowing the model to effectively approximate the prediction behavior of the server-side back end while remaining suitable for deployment at resource-constrained local nodes. As described in Section IV-C, the trained lightweight model is deployed at each local node and operates jointly with the front-end model to enable independent intrusion detection.

Table III summarizes the architectural configurations of the global full model and the lightweight student model. Based on the above model configurations, we conduct a series of ablation studies to systematically analyze the impact of key design factors, including the cut-layer position and the distillation strategy.

D. Ablation Study

The proposed framework integrates multiple mechanisms, including split-based distributed learning, online KD, and a shadowing process designed to maintain functional alignment between the global full model and the lightweight model. These components operate in a tightly coupled manner, and design configurations associated with each mechanism can significantly influence training efficiency, resource consumption, and intrusion detection accuracy. Accordingly, we aim to systematically investigate how key architectural and algorithmic components of the proposed SFL-KD framework affect both detection performance and operational efficiency. To quantify the contribution of individual components and structural parameters, we conduct a comprehensive ablation analysis. In particular, from a distributed learning perspective, we investigate how different cut-layer positions affect the computational workload distribution between local nodes and the central server and the associated communication overhead during training. In addition, the effects of KD and the shadowing process are analyzed in terms of their influence on predictive performance consistency between the global full model and the deployed lightweight detector.[7]

1) Cut-Layer Analysis: Since the proposed system adopts a split-based learning architecture, the selection of the cut layer directly affects multiple aspects of the system, including the computational workload at local nodes, the communication overhead, and the convergence behavior of the learning process. Consequently, the cut-layer configuration plays a critical role in balancing system efficiency and intrusion detection performance. To this end, we perform a detailed cut-layer analysis to investigate the effects of different partition points within the global model. As described earlier, the global intrusion detection model is structured as an eight-layer network. Based on this architectural configuration, we restrict the cut-layer candidates to the first three layers of the model. Note that this restriction is based on the practical assumption that local nodes operate under limited computational resources and are, therefore, assigned a shallower portion of the overall model than the central server.

Table IV reports the local computational cost incurred at each cut-layer configuration in terms of training and inference FLOPs, measured per minibatch with a batch size of 256. As expected, the training cost at local nodes increases monotonically as the cut layer moves deeper into the detection model. This trend stems from the expanded front-end network executed at local nodes, which results in higher forward and backward computation. Specifically, the total training FLOPs increase from 35.8M at cut layer 1 to 86.2M at cut layer 2, and further to 213.4M at cut layer 3. This growth reflects the inclusion of additional convolutional and fully connected layers in the local front end, which significantly amplifies the backward computation cost during training. In contrast, the inference cost exhibits a more pronounced increase at deeper cut layers. While the inference FLOPs remain relatively stable

[7]All ablation experiments are conducted on the 5G-NIDD dataset under a non-IID data distribution scenario, with the batch size set to 256 for each training iteration.

TABLE IV
LOCAL COMPUTATIONAL COST BY CUT LAYER (BATCH SIZE = 256)

Cut-layer	Train Fwd FLOPs	Train Bwd FLOPs	Train Total FLOPs	Infer Total FLOPs
1	11,927,552	23,920,640	35,848,192	570,163,200
2	28,704,768	57,507,840	86,212,608	578,551,808
3	69,337,088	144,146,432	213,483,520	1,654,882,304

TABLE V
COMMUNICATION OVERHEAD PER GLOBAL EPOCH BY CUT LAYER (PER LOCAL NODE)

Cut-layer	Feat. Elems	Front Params	Split/Epoch	FedAvg/Epoch	Total/Epoch
1	256	23,552	232.5 MB	0.18 MB	232.7 MB
2	128	56,448	118.74 MB	0.43 MB	119.17 MB
3	15,872	57,216	14.37 GB	0.44 MB	14.38 GB

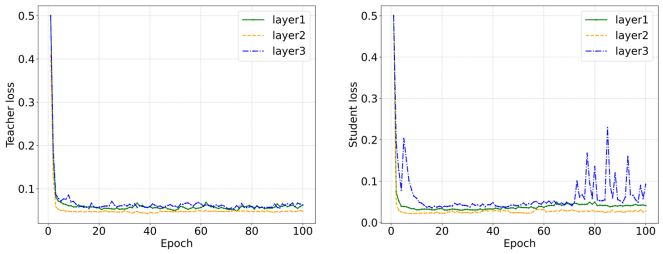

Fig. 2. Training convergence behavior of the teacher (global full) and student (lightweight) models under different cut-layer configurations.

between cut layer 1 and cut layer 2, a substantial escalation is observed at cut layer 3. This behavior is primarily attributed to the enlarged smashed representation generated at deeper partition points, which directly increases the computational burden of the fixed student model during inference. A similar trend is also observed in terms of communication overhead, as summarized in Table V. In a split-based learning framework, the dominant source of communication cost arises from the bidirectional transmission of smashed representations and their corresponding gradients between local nodes and the central server. The magnitude of this overhead is determined by the dimensionality of the smashed representation generated at the cut layer, which can vary significantly depending on the architectural characteristics of the partition point. At cut layer 1, the communication overhead per local node remains relatively moderate since the smashed representation consists of only 256 feature elements, resulting in approximately 232.7 MB of data transfer per global epoch. Cut layer 2 further reduces the communication cost, as the projection layers compress the intermediate representation to 128 features, yielding the lowest split communication overhead. In contrast, cut layer 3 leads to a drastic increase in communication overhead, reaching 14.38 GB per global epoch per local node. This sharp escalation is caused by the high-dimensional smashed representation generated after the convolutional layers. Note that the communication cost associated with the aggregation process remains minor across all cut-layer configurations, as only the front-end model parameters are exchanged.

Beyond the operational efficiency aspects, the cut-layer configuration is also closely related to the convergence behavior of the intrusion detection models. Specifically, the cut layer determines the characteristics of the intermediate (smashed) representations, which, in turn, influence the stability of the training dynamics. As illustrated in Fig. 2, the teacher models corresponding to different cut-layer configurations exhibit comparable convergence trends, suggesting that the back end can consistently learn discriminative patterns and is largely insensitive to the cut-layer configuration. In contrast, the convergence behavior of the lightweight student model shows more noticeable differences across cut-layer configurations. For cut layer 1 and cut layer 2, the student loss decreases smoothly and stabilizes at relatively low values, indicating effective knowledge transfer from the teacher model. In particular, cut layer 2 demonstrates the most stable convergence profile, with consistently lower loss and reduced oscillation throughout training. By comparison, cut layer 3 exhibits a markedly less stable convergence pattern for the student model, characterized by frequent loss spikes and increased variance. This instability can be attributed to the high-dimensional smashed representations produced after the convolutional layers, which significantly increase the complexity of the input space to the student model and hinder consistent KD.

Based on these observations, the selection of the cut layer can be guided by jointly considering the local computational budget, the available communication bandwidth, and the stability of KD. When local nodes operate under stringent computational constraints (e.g., limited CPU/GPU capacity or concurrent execution of other local processes), a shallower cut layer (e.g., cut layer 1) is preferable, as it minimizes the local training and inference FLOPs at the cost of moderately higher communication overhead. Conversely, when communication bandwidth is the primary bottleneck, a deeper cut layer (e.g., cut layer 2) can substantially reduce the volume of transmitted smashed data, as the projection layers compress the intermediate representation into a lower-dimensional space. However, excessively deep cut layers (e.g., cut layer 3) should be avoided, as they not only incur a drastic increase in both communication overhead and inference cost but also lead to unstable KD due to the high-dimensional smashed representations. Taking these factors into account, the experimental results indicate that cut layer 2 offers the most balanced configuration under the proposed framework, effectively avoiding the excessive local computational burden

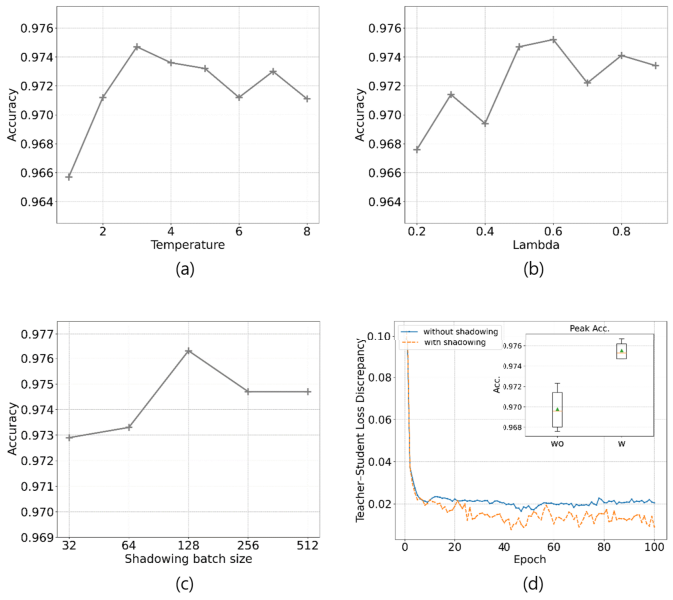

Fig. 3. Ablation results for the distillation and shadowing components. (a) Detection accuracy under different temperature values. (b) Impact of the loss balancing coefficient λ on detection performance. (c) Effect of the shadowing batch size on detection accuracy. (d) Comparison of teacher–student loss discrepancy during training with and without shadowing.

while maintaining lower communication overhead and stable convergence for both the back-end training and the KD process.

2) Distillation Hyperparameter Sensitivity and Shadowing Analysis: Building upon the cut-layer analysis, we examine the effects of KD and shadowing on the stability and effectiveness of the proposed framework. While online distillation enables the transfer of discriminative knowledge from the server-side back end to the lightweight student model, its performance is inherently influenced by the characteristics of the distillation process. In particular, the quality and consistency of the distillation signal depend on the configuration of hyperparameters and the temporal variability of the teacher logit distribution. To provide a comprehensive understanding of the factors influencing distillation stability, we first analyze the sensitivity of the distillation process to key hyperparameters, including the temperature T and loss balancing coefficient λ. We then analyze how shadowing complements the distillation process by reducing the sensitivity of the student model to the temporal variability of the teacher logit distribution during training.

Fig. 3 illustrates the ablation results for the distillation process. As shown in Fig. 3(a) (distillation sensitivity to temperature scaling), the detection model achieves its highest accuracy at a moderate temperature value of $T = 3$, whereas both lower and higher temperature settings lead to performance degradation. This trend reflects an inherent tradeoff in temperature scaling, where lower temperatures maintain sharp teacher predictions but constrain the propagation of interclass similarity information, while higher temperatures introduce excessive logit smoothing and reduce the discriminative quality of the distilled knowledge. From the perspective of the loss balancing coefficient λ, which controls the relative weighting between the supervised loss and the distillation loss, the intrusion detection model achieves its highest accuracy at $\lambda = 0.6$, as shown in Fig. 3(b) (tradeoff between ground-truth supervision and teacher guidance). This observation indicates that the lightweight student model exhibits the best detection performance when it is trained with a slightly higher emphasis on the teacher-provided distillation signal. The effectiveness of the distillation process remains sensitive to the temporal variability of the teacher outputs during training, which motivates the introduction of shadowing in the following analysis.

Rather than relying on instantaneous teacher logits, shadowing maintains a buffered representation of teacher outputs aggregated over multiple minibatches. This mechanism provides a more stable distillation signal to the student model by reducing abrupt fluctuations in the distillation targets. Under this design, the shadowing batch size determines the temporal scope over which teacher outputs are accumulated, which directly influences the stability–adaptability tradeoff of the distillation process. Fig. 3(c) (stability–adaptability tradeoff of shadowing) reports the detection accuracy of the student model under different shadowing batch sizes. As the shadowing batch size increases from 32 to 128, the detection accuracy improves and reaches its maximum at a batch size of 128. This behavior suggests that aggregating teacher outputs over a moderate temporal window effectively suppresses short-term noise in the distillation signal. When the shadowing batch size is further increased, the performance gain diminishes and slightly degrades, indicating that excessively large aggregation windows may delay the adaptation of the student model to evolving teacher outputs.

The effectiveness of shadowing is further illustrated in Fig. 3(d), which compares the teacher–student loss discrepancy during training with and without shadowing. Note that, to compute the teacher–student loss discrepancy, we separately measure the classification loss (i.e., cross-entropy loss) of the student model at each training round. When shadowing is applied, the loss discrepancy stays at a consistently lower level throughout training. This lower loss discrepancy is accompanied by a higher peak detection accuracy, as shown in the inset of Fig. 3(d) (effect of shadowing on the teacher–student gap). In contrast, without shadowing, the teacher–student loss discrepancy remains at a higher level throughout training, which indicates an increased sensitivity of the distillation process to the teacher's behavior.

E. Performance Evaluations

To assess both the detection performance and resource efficiency of the proposed framework, we conduct a series of experimental evaluations. The evaluation focuses on verifying whether the proposed framework can achieve competitive intrusion detection performance while significantly reducing the computational overhead on local nodes. We compare the proposed approach with several baseline models, including: 1) a centralized detection model trained on the full dataset; 2) a widely adopted FL setup; and 3) an SL configuration. To provide a more comprehensive comparison across different distributed learning strategies, we further consider representative FL variants, including: 4) FedProx [36] for client

TABLE VI
ACCURACY AND PER-CLASS $F1$-SCORE COMPARISON ON THE OPEN RAN DATASET

Scenario	Algorithm	Accuracy	Normal				Attack			
			Web Browsing	VoIP	IoT Messaging	Streaming	DDoS-Ripper	DoS-Hulk	PortScan	Slowloris
–	Centralized	0.9260	0.9029	0.9649	0.9856	0.9612	0.8480	0.8969	0.9452	0.8954
IID	FL	0.9156	0.8840	0.9558	0.9821	0.9562	0.8318	0.8856	0.9383	0.8789
	SL	0.9171	0.8878	0.9588	0.9812	0.9565	0.8315	0.8869	0.9402	0.8814
	FedProx	0.9169	0.8873	0.9563	0.9816	0.9561	0.8326	0.8870	0.9394	0.8819
	SCAFFOLD	0.9214	0.8975	0.9582	0.9821	0.9612	0.8437	0.8933	0.9417	0.8842
	FedOpt	0.9170	0.8884	0.9573	0.9829	0.9560	0.8314	0.8867	0.9399	0.8811
	SFL-KD (ours)	0.9222	0.8965	0.9620	0.9819	0.9597	0.8418	0.8932	0.9450	0.8862
Non-IID	FL	0.9104	0.8714	0.9531	0.9796	0.9500	0.8235	0.8820	0.9350	0.8692
	SL	0.9062	0.8601	0.9496	0.9761	0.9506	0.8175	0.8789	0.9336	0.8578
	FedProx	0.8999	0.8469	0.9475	0.9742	0.9431	0.8127	0.8756	0.9264	0.8384
	SCAFFOLD	0.8988	0.8527	0.9453	0.9633	0.9485	0.8019	0.8766	0.9300	0.8278
	FedOpt	0.9046	0.8606	0.9487	0.9795	0.9500	0.8124	0.8810	0.9274	0.8419
	SFL-KD (ours)	0.9187	0.8927	0.9544	0.9822	0.9582	0.8365	0.8901	0.9412	0.8781

drift mitigation; 5) SCAFFOLD [42] for control variate-based correction; and 6) FedOpt implemented using the FedAdam optimizer [43] with server-side adaptive optimization. Classification performance is evaluated using two standard metrics: accuracy and $F1$-score (the harmonic mean of precision and recall). Furthermore, to quantify the efficiency of each detection model, we measure resource consumption by evaluating computational cost and inference latency.

1) Evaluation on Open RAN Dataset: The dataset consists of control and user-plane signaling traces captured in an Open RAN (Open Radio Access Network) environment. To reflect the characteristics of an Open RAN environment, we assume a distributed setting consisting of multiple open distributed units (O-DUs) and a centralized O-CU (Open Central Unit) serving as the central server. Each O-DU is regarded as a resource-constrained node, where minimizing computational overhead is crucial, as most of the available resources are dedicated to radio signal processing. Table VI presents the experimental results for each data distribution scenario.

In the IID setting, all distributed learning approaches show strong detection performance, while the centralized model achieves an accuracy of 0.9260. The proposed SFL-KD framework achieves an accuracy of 0.9222, resulting in only a marginal performance gap compared to the centralized baseline. In comparison, the FL- and SL-based models achieve accuracies of 0.9156 and 0.9171, respectively, indicating that they maintain acceptable detection performance while exhibiting a slight performance degradation. Among the FL-based variants, FedProx, SCAFFOLD, and FedOpt achieve accuracies of 0.9169, 0.9214, and 0.9170, respectively, all of which outperform the standard FL baseline while remaining slightly below the proposed SFL-KD framework. This observation indicates that incorporating advanced optimization and drift-mitigation mechanisms improves detection performance. A closer inspection of the per-class $F1$-scores in Table VI shows that SFL-KD maintains consistently strong performance across both normal traffic types and attack categories, particularly for PortScan and DDoS-related attacks, where its performance closely matches that of the centralized model.

In the non-IID scenario, imbalanced local data distributions disrupt consistent optimization across local nodes, which leads to a more pronounced performance gap among learning approaches. Compared to the IID setting, conventional distributed learning methods exhibit a visible degradation in detection performance. The FL-based model achieves an accuracy of 0.9104, indicating that heterogeneous data distributions across local nodes undermine stable global optimization. Since each local node computes updates based on biased local data, the aggregated gradients tend to diverge across training rounds. A similar degradation trend is observed for other FL-based variants. FedProx and FedOpt achieve accuracies of 0.8999 and 0.9046, respectively, while SCAFFOLD further drops to 0.8988. Although these methods introduce additional mechanisms for mitigating client drift or stabilizing optimization, their performance remains sensitive to severe data heterogeneity, suggesting that optimization-level algorithmic corrections are insufficient to fully address non-IID effects. The SL framework shows further degradation, with an accuracy of 0.9062, which can be attributed to its sequential training paradigm. In particular, the relay-based update process amplifies the influence of locally biased data and leads to unstable convergence behavior across successive nodes. In contrast, the proposed SFL-KD framework achieves an accuracy of 0.9187, outperforming all baseline methods under the non-IID setting. This performance gain stems from the hybrid learning structure that combines parallel local updates with a centralized back-end optimization process. In particular, the back-end model is configured with

TABLE VII
ACCURACY AND PER-CLASS $F1$-SCORE COMPARISON ON THE 5G-NIDD DATASET

Scenario	Algorithm	Accuracy	Normal	Attack							
			Benign	HTTP Flood	ICMP Flood	SYN Flood	SYN Scan	Slowloris	TCP Scan	UDP Flood	UDP Scan
–	Centralized	0.9830	0.9789	0.9973	1.0000	0.9982	0.9979	0.9947	0.9988	0.9787	0.9984
IID	FL	0.9823	0.9782	0.9969	0.9979	0.9984	0.9973	0.9940	0.9985	0.9780	0.9978
	SL	0.9825	0.9783	0.9971	1.0000	0.9984	0.9977	0.9945	0.9982	0.9781	0.9978
	FedProx	0.9822	0.9782	0.9965	0.9958	0.9979	0.9977	0.9933	0.9989	0.9779	0.9980
	SCAFFOLD	0.9826	0.9786	0.9965	1.0000	0.9979	0.9972	0.9934	0.9989	0.9784	0.9977
	FedOpt	0.9824	0.9784	0.9967	1.0000	0.9982	0.9974	0.9939	0.9986	0.9782	0.9981
	SFL-KD (ours)	0.9824	0.9783	0.9971	1.0000	0.9984	0.9979	0.9944	0.9986	0.9780	0.9973
Skewed	FL	0.9419	0.9444	0.9642	0.9119	0.9087	0.8827	0.9440	0.9519	0.9433	0.6136
	SL	0.9620	0.9589	0.9733	0.8293	0.9029	0.9923	0.9575	0.9454	0.9638	0.9516
	FedProx	0.9006	0.9136	0.9196	-	0.3570	0.8960	0.8622	0.7395	0.9194	-
	SCAFFOLD	0.9432	0.9668	0.9868	-	0.9100	0.5453	0.9743	0.3525	0.9668	-
	FedOpt	0.9563	0.9623	0.9523	0.9655	0.7817	0.9391	0.9036	0.9554	0.9636	0.9610
	SFL-KD (ours)	0.9770	0.9717	0.9959	0.9979	0.9979	0.9977	0.9922	0.9969	0.9714	0.9973

a more expressive architecture to learn informative representations over the projected feature space. Through online KD, the lightweight detection model is trained to align with the soft output distribution of the back-end model, which allows it to inherit the decision behavior of the high-capacity server-side model. This alignment mechanism reduces the discrepancy induced by heterogeneous local updates and leads to more stable training dynamics. As a result, the proposed framework maintains superior detection accuracy and improved training stability compared to conventional FL and SL approaches in non-IID environments (for further details, aggregate performance metrics and per-class confusion matrices for each method under the non-IID scenario are provided in Appendix A).

2) Evaluation on 5G-NIDD Dataset: To further assess the effectiveness of the proposed framework in a realistic distributed mobile network environment, we conduct additional experiments on the 5G-NIDD dataset. To emulate a distributed deployment, we assume a topology where multiple local nodes are located at distributed 5G base stations to proactively analyze the network traffic, while a centralized MEC server supports global model training and overall coordination. Note that each local node is assumed to operate under computational constraints, as its primary function is to process high-throughput network flows generated by the 5G radio stack. Table VII summarizes the experimental results in terms of detection accuracy and classwise performance.

In the IID setting, all distributed learning approaches achieve detection performance comparable to the centralized model, which is consistent with the trends observed in the Open RAN experiments. This result indicates that distributed learning approaches perform effectively when local data distributions closely reflect an ideal setting. In contrast, the skewed scenario presents a more severe challenge for distributed training. In this setting, each local node is assigned the complete set of data for a single attack class along with a portion of benign traffic, leading to highly biased local updates and disrupting stable global optimization. Under this severely imbalanced condition, conventional distributed learning approaches exhibit a pronounced degradation in detection performance. The FL-based model achieves an accuracy of 0.9419, while the SL-based model reaches 0.9620, reflecting their limited ability to generalize under severely imbalanced local data distributions. Similar behavior is observed for the advanced FL variants although their performance varies noticeably across algorithms. FedProx exhibits a substantial performance drop, achieving an overall accuracy of 0.9006, with several attack categories showing severe degradation or even missing predictions due to unstable training. SCAFFOLD demonstrates slightly improved robustness compared to FedProx, reaching an accuracy of 0.9432, yet it still suffers from pronounced performance degradation for specific attack types, particularly those with limited sample availability. FedOpt attains a relatively higher accuracy of 0.9563, indicating improved stability in comparison to other FL variants. However, its performance remains inconsistent across attack categories, with noticeable degradation observed for certain rare attacks. During training, these FL-based approaches frequently exhibit unstable optimization dynamics under the skewed setting, characterized by abrupt loss spikes and difficulty in achieving stable convergence. Such instability becomes particularly evident for attack types with extremely limited data volume. For instance, ICMP Flood and related rare attack classes are often not effectively learned by several baseline methods, as reflected by missing or severely degraded per-class $F1$-scores in Table VII. This behavior can be attributed to the dominance of benign traffic samples that are inherently included at each local node. During optimization,

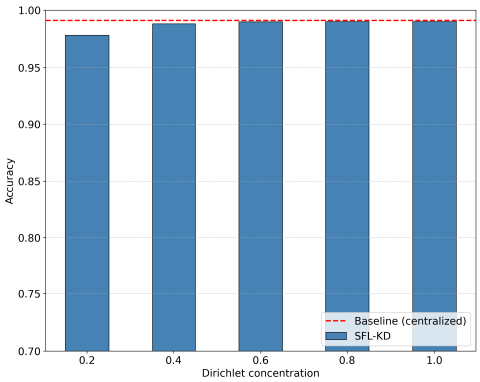

Fig. 4. Accuracy comparison between the proposed SFL-KD framework and the centralized baseline on an IoT dataset across varying degrees of data heterogeneity.

gradient dominance from the benign class tends to outweigh that of rare attack samples, which suppresses the learning signal associated with minority attack patterns and reduces their discriminative contribution. In contrast, the proposed SFL-KD framework maintains strong detection performance, achieving an accuracy of 0.9770, which closely approaches the centralized baseline. This performance advantage stems from the hybrid learning design that combines parallel local front-end updates with centralized back-end optimization. By learning expressive representations at the server side and transferring them to the lightweight detection model through online KD, SFL-KD alleviates the bias induced by locally skewed data and improves generalization. As a result, the proposed framework preserves high detection accuracy and stable performance under extreme class imbalance, demonstrating its effectiveness in realistic and highly constrained intrusion detection environments (for further details, aggregate performance metrics and per-class confusion matrices for each method under the skewed scenario are provided in Appendix B).

3) Generalizability to IoT Environments: To further evaluate the generalizability of the proposed framework to IoT environments, we conduct additional experiments on the CIC-IoT-Dataset-2023 [44]. The dataset contains 33 distinct attack types organized into seven categories—DDoS, DoS, Reconnaissance, Web-based, Brute Force, Spoofing, and Mirai—along with benign traffic. Each record consists of 47 flow-level features capturing protocol behavior, packet statistics, and timing characteristics. In this experiment, we adopt the same preprocessing pipeline and model configuration described in Sections IV-A and V-C. To assess robustness under heterogeneous data distributions, we evaluate the proposed framework across varying degrees of non-IID partitioning (using a Dirichlet distribution with concentration parameters $\alpha \in \{0.2, 0.4, 0.6, 0.8\}$). Fig. 4 presents the detection performance of the proposed SFL-KD framework, along with the centralized baseline for reference. As shown in the figure, the proposed SFL-KD framework consistently achieves detection accuracy closely approaching the centralized baseline (which is trained on the full dataset, achieving an accuracy of 0.9912). Under relatively mild heterogeneity settings

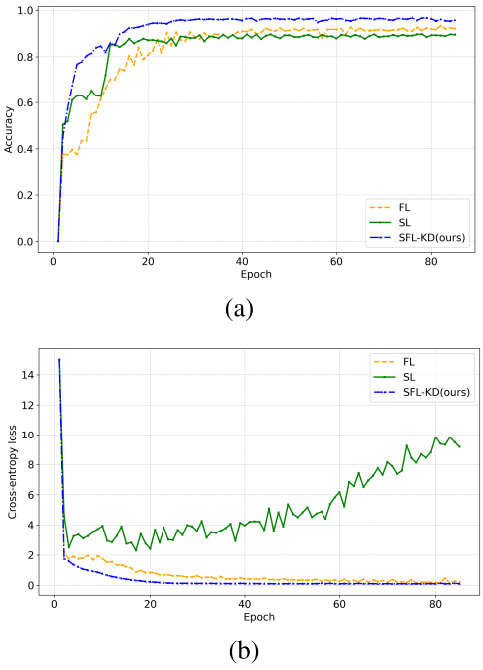

Fig. 5. Comparison of test performance under the data imbalance scenario: (a) test accuracy and (b) cross-entropy loss on the test data, measured at each epoch.

(i.e., $\alpha \geq 0.6$), SFL-KD achieves an accuracy above 0.9899, yielding a marginal difference of approximately 0.1% from the centralized model. At $\alpha = 0.4$, SFL-KD attains an accuracy of 0.9882, still remaining close to the centralized baseline. Even under a highly heterogeneous condition ($\alpha = 0.2$), SFL-KD maintains an accuracy of 0.9782, resulting in only approximately 1.3 percentage points of performance gap. This trend is consistent with the observations from the 5G-NIDD and Open RAN experiments, where the proposed framework maintained competitive detection performance under heterogeneous data distributions. Note that, under the highly heterogeneous condition (i.e., $\alpha = 0.2$), conventional distributed learning approaches such as FL, SL, and FedOpt achieved accuracies of 0.9615, 0.9544, and 0.9627, respectively, exhibiting more than 2.8 percentage points of performance degradation compared to the centralized model. These results further confirm that the proposed framework generalizes effectively to IoT network environments, preserving robust detection capability even under pronounced data heterogeneity.

F. Convergence, Efficiency, and Real-Time Performance

1) Training Convergence Behavior: To analyze the training stability of the proposed framework, we compare its convergence behavior against other distributed learning approaches under the imbalanced data distribution scenario using the 5G-NIDD dataset. Fig. 5 depicts the loss variations and accuracy trends on the test set, measured at each global training epoch for each learning approach. Note that although the student model in the proposed framework is optimized using the KD loss, we additionally measured the standard classification loss (i.e., cross-entropy) at each epoch as an auxiliary metric to analyze convergence behavior.

As previously demonstrated, all learning frameworks achieve satisfactory performance under the ideal data distribution, with their training processes showing stable convergence. However, under the data imbalance scenario, the training behavior diverges notably across the frameworks, as shown in Fig. 5. In the case of FL, convergence is still attainable, but the accuracy curve shows delayed stabilization (requiring approximately 40 epochs to reach a stable accuracy level, with noticeable oscillations of up to 3 percentage points throughout the training process). These symptoms indicate that local updates from biased class distributions result in inconsistent aggregation, which, in turn, leads to delayed convergence.[8] The irregularities become more pronounced in SL, where the relay-based training across nodes amplifies instability. Specifically, SL tends to exhibit increased loss at node transitions, which arise from abrupt shifts in class distribution between local nodes. Moreover, it was observed that the test accuracy remained nearly constant (around 90%), while the test loss gradually increased as training progressed, indicating a growing discrepancy between training and generalization performance. These patterns suggest that the global model in SL becomes increasingly overfitted to the restricted class distribution of the current node, leading to degraded generalization and resulting in repeated loss spikes. In contrast, the proposed SFL-KD framework demonstrates a stable convergence pattern despite the presence of imbalanced and heterogeneous class distributions. Specifically, the test accuracy reaches above 97% within approximately 15 epochs and remains stable with fluctuations of less than 1 percentage point throughout the remaining training period. This convergence behavior is reflected in the experimental results, where the test accuracy quickly saturates above 97%, while the test loss consistently decreases without significant fluctuation. This stability can be attributed to the richer representational capacity of the back-end model and the integration of KD, which facilitates the transfer of informative representations to the lightweight model. Building on this systematic integration, the proposed framework effectively preserves high-level representations throughout training and promotes stable convergence. It is worth noting that under the extreme skewed distribution scenario, the divergence between learning frameworks becomes more pronounced—SL frequently fails to converge, and FL often reaches a saturation point early in training (see [45] for more detailed discussions on convergence analysis in distributed learning settings).

2) Local Resource Utilization: To assess the computational efficiency in the training phase, we measure the resource consumption incurred at local nodes using the 5G dataset. This evaluation is particularly relevant for practical application in distributed environments, where local devices typically operate under stringent resource constraints.

[8]Note that, although representative FL variants (i.e., FedProx, SCAFFOLD, and FedOpt) were also evaluated under the same condition, their convergence patterns were found to be highly similar to that of the standard FL baseline. Therefore, for clarity of visualization, we present only the FL curve as a representative of the FL-family methods in Fig. 5, avoiding visual overlap that could hinder readability.

TABLE VIII
NUMBER OF TRAINABLE PARAMETERS ASSIGNED TO EACH LOCAL NODE

Approach	Assigned Module	Param Count	Weight (%)
Centralized	Front-end + back-end	153,449	100.0
FL	Front-end + back-end	153,449	100.0
SL	Front-end only	56,448	36.8
SFL-KD (ours)	Front-end only	56,448	36.8

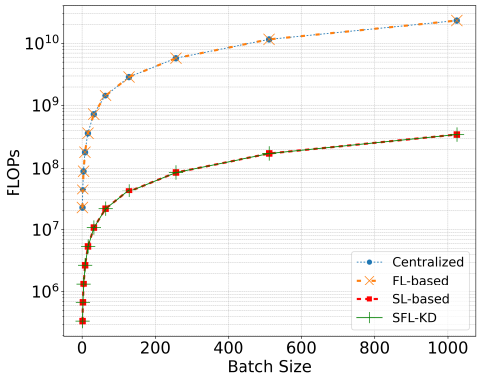

Fig. 6. Comparison of local FLOPs over batch sizes.

As described above, the practical detection model operated at each node is structured as a composite of a front-end feature extractor and a lightweight student classifier. In the FL framework, the entire model is trained locally at each node. In both the SL and proposed SFL-KD frameworks, local nodes handle only the front-end model during training, while the classification component is trained centrally. Based on this structural distinction, we can estimate the model size assigned to local nodes for each learning framework. Given the input dimension and layer configuration of the detection model, the number of trainable parameters allocated to each local node can be derived analytically. Table VIII summarizes the local model size in terms of the number of parameters for each framework. Note that representative FL variants such as FedProx, SCAFFOLD, and FedOpt share the same model architecture as the standard FL setting, where the entire detection model is trained locally at each node. As shown in the table, the FL configuration assigns approximately 153 449 trainable parameters to each local node, meaning that each local node is required to train the full model independently. Compared to FL, SL and the proposed SFL-KD assign only 56 448 parameters to each node, as only the front-end model is subject to local training. This architectural difference results in an analytical reduction of 63.2% in local model size compared to the FL setting.

To quantify the computational burden at local nodes, we measured the number of floating-point operations (FLOPs) required for local model processing under varying batch sizes from 1 to 1024. Fig. 6 presents the measurement results. In the FL-based setting, where the full model is executed locally as in the centralized baseline, FLOPs amounted to 23, 150, 362, and 624 at a batch size of 1024. In contrast, SL and the proposed SFL-KD process only the front-end model at local

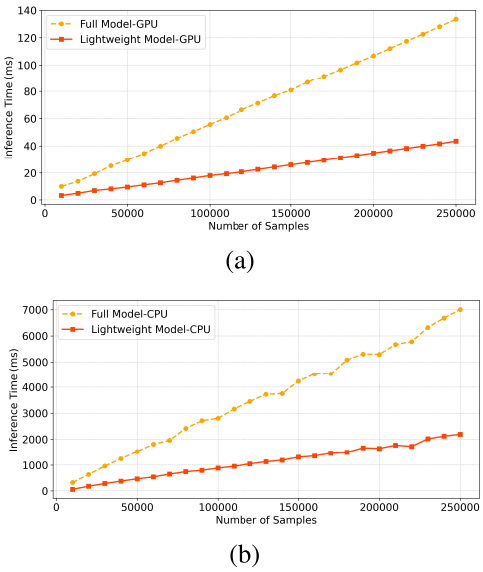

Fig. 7. Comparison of local inference time on (a) GPU and (b) CPU.

nodes, with FLOPs measured at 344, 850, and 432 at the same batch size. Across all batch sizes, SL and SFL-KD consistently exhibit a markedly smaller computational footprint than the FL and centralized settings. These findings demonstrate that the proposed framework (like SL) can provide substantial computational efficiency at the node level.

3) Operational Performance in Real-Time Detection: To enable proactive responses to potential threats at edge locations, real-time capability is a crucial requirement for NIDSs. In next-generation and industrial networks, latency-sensitive scenarios require agile detection modules. Therefore, evaluating how each architectural design affects real-time processing performance is important for assessing its deployability under real-world conditions. From this perspective, we conducted empirical measurements to compare the inference throughput of two model configurations: a centralized full model consisting of both front- and back-end modules, and a lightweight model comprising the front-end and student classifier. The former reflects the structure commonly adopted in centralized intrusion detection systems, where high detection performance is prioritized at the cost of computational complexity. In contrast, the lightweight configuration is designed to support efficient and timely detection in distributed environments, particularly at resource-constrained edge locations. In such conditions, the key challenge is to design lightweight detection models that improve computational efficiency while maintaining detection performance. To compare the operational efficiency, we measure the time taken to process chunks of data ranging from 10 000 to 250 000 samples on the 5G dataset. Note that the experiment focuses solely on the inference step (i.e., forward pass).

Fig. 7 shows the inference time as the number of input samples increases. In the GPU-based environment [see Fig. 7(a)], the full model exhibited a rapid increase in latency, requiring over 0.13 s to process 250 000 samples. In contrast, the lightweight model completes the same inference task in approximately 0.043 s, achieving roughly a 3.1× improvement in inference time. This latency reduction becomes more prominent as the input size increases, confirming that the simplified architecture enables significantly faster inference under identical conditions. A similar pattern is observed in the CPU-based evaluation [see Fig. 7(b)], which reflects resource-limited scenarios where GPU acceleration is not available. The latency of the full model exceeds 7 s at 250 000 samples, whereas the lightweight model completes inference in approximately 2.2 s. Although both environmental configurations exhibit linear growth in inference time, the absolute delay incurred by the full model renders it infeasible for real-time intrusion detection in nonaccelerated environments. These results collectively demonstrate that architectural simplification can substantially enhance inference efficiency by reducing computational complexity. That is, the lightweight configuration can support timely responses under constrained conditions, making it a practical solution for real-time deployment in next-generation distributed networks. It is important to note that, as previously analyzed, such improvements are achieved at the cost of slight performance tradeoffs. Nevertheless, the tradeoff remains relatively minor and can be acceptable in scenarios where low latency and rapid response are prioritized. Moreover, the lightweight model employed in our SFL-KD framework still demonstrated competitive detection performance compared to other distributed learning frameworks. These findings confirm that the proposed SFL-KD framework enables effective and efficient real-time intrusion detection in resource-constrained distributed networks, achieving a balanced tradeoff between detection performance and computational feasibility.

To further assess the practical applicability under realistic edge deployment conditions, we additionally measured the inference latency at smaller batch sizes, as summarized in Table IX. At single-flow granularity (data size = 1), the lightweight model processes an individual network flow in approximately 0.14 ms on GPU and 1.48 ms on CPU, compared to 0.30 and 1.83 ms for the full model, respectively. As the batch size increases, the latency gap between the two configurations becomes more pronounced, particularly in CPU environments, where the lightweight model achieves up to 2.9× faster processing at a batch size of 1000. These granular measurements confirm that the lightweight model can support flow-level real-time detection with submillisecond latency on GPU, demonstrating its feasibility for latency-sensitive edge deployment scenarios.

4) Stability Under Communication Dynamics: Throughout our experiments, we assumed a fixed 20% node dropout per round to reflect realistic network conditions, in which a fraction of local nodes becomes temporarily unavailable. This setting, however, captures only total connectivity loss and not the latency variability that arises when a node remains connected yet its communication is delayed by various factors—fading, jitter, bandwidth fluctuation, or congestion. To comprehensively capture these finer-grained dynamics of realistic mobile networks, we evaluate the stability of SFL-KD under stochastic transport latency. To model these conditions, we simulate the end-to-end latency of each per-batch transmission—the smashed data on the uplink (UL) and the

TABLE IX
INFERENCE LATENCY UNDER SMALL BATCH SIZES ON GPU AND CPU

Data Size	GPU (ms)		CPU (ms)	
	Full Model	Lightweight	Full Model	Lightweight
1	0.2998 ± 0.2560	0.1414 ± 0.1628	1.8315 ± 0.5106	1.4849 ± 0.2122
10	0.2982 ± 0.2572	0.1427 ± 0.1650	1.3287 ± 0.2522	0.8400 ± 0.0526
50	0.4900 ± 0.0657	0.3182 ± 0.0099	2.0386 ± 0.2481	1.0165 ± 0.0364
100	0.5208 ± 0.5779	0.3485 ± 0.4547	2.9481 ± 0.0562	1.4311 ± 0.0458
500	0.5126 ± 0.5694	0.3522 ± 0.4066	12.9720 ± 1.3448	4.7090 ± 0.2220
1000	0.5996 ± 0.5585	0.3678 ± 0.4290	23.4030 ± 1.2058	8.0447 ± 0.1478

cut-layer gradient on the downlink (DL)—as an independent log-normal variable, $\ell_{\mathrm{UL}}, \ell_{\mathrm{DL}} \sim \mathrm{LogNormal}(\mu, \sigma)$.[9] In SFL-KD, we adopt a deadline-bounded synchronous mechanism with straggler drop, without waiting for delayed nodes. At each minibatch, only nodes whose latency is within a deadline τ contribute to the model updates, whereas those exceeding τ are dropped from that batch alone and reattempt in subsequent batches. Since the server never waits beyond τ, delayed nodes cannot stall training, which continues at the cost of their occasional missed updates. Building on this mechanism, we assess the stability of SFL-KD by generating stochastic per-batch latency that jointly captures communication dynamics.

To quantify the stability, we first calibrate the deadline τ via an in-training measurement, rather than a static or worst case estimate. By running the training process for 5 epochs at a representative jitter level ($\sigma = 0.2$) without enforcing any deadline, we obtain the empirical observations of per-batch end-to-end latency across all nodes and minibatches. We then set τ to the 90th percentile of these observations. Note that P90 strikes a balance between overly aggressive dropping (e.g., P50) and effectively nonbinding deadlines (e.g., P99), thereby enforcing meaningful straggler drop while retaining sufficient node participation. With this calibrated τ, we sweep the jitter intensity $\sigma \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ to evaluate the stability of the proposed framework, training each configuration on 5G-NIDD with non-IID Dirichlet partitioning ($\alpha = 0.3$). Table X reports two communication-level metrics—per-node drop ratio and full-participation ratio—alongside the test accuracy. These metrics together characterize the extent of per-batch communication loss arising from various transport conditions. As σ increases, the network conditions monotonically degrade across both metrics, with the per-node drop ratio rising from 0.0% to 49.3% and the full-participation ratio dropping from 100.0% to 24.0%. Despite this substantial communication degradation, however, the test accuracy remains nearly invariant, as shown in Fig. 8, dropping only 0.64 percentage points from 97.68% ($\sigma = 0$) to 97.04% ($\sigma = 1.0$). Quantitatively, a fivefold increase in per-node drop ratio—from 9.7% at $\sigma = 0.2$ to 49.3% at $\sigma = 1.0$—produces less than 1 percentage point of accuracy degradation, demonstrating that SFL-KD can preserve its performance even under such substantial per-batch transport instability.

[9]Here, $\exp(\mu)$ and σ denote the median one-way latency and the jitter intensity, respectively. To reflect typical mobile network conditions, we set $\exp(\mu) = 50$ ms and truncate sampled latencies at 2000 ms to bound the log-normal tail.

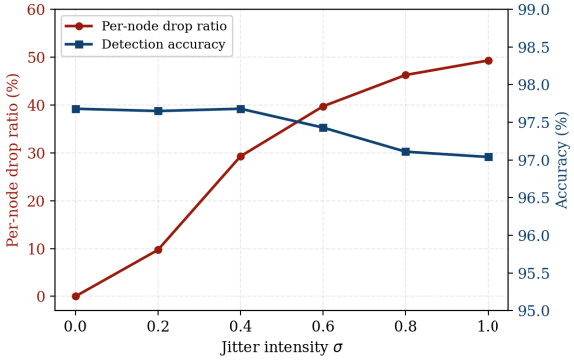

Fig. 8. Per-node drop ratio (left axis, red) and student detection accuracy (right axis, blue) as a function of jitter intensity σ.

TABLE X
STABILITY OF SFL-KD UNDER INCREASING JITTER INTENSITY σ (NON-IID DIRICHLET $\alpha = 0.3$)

σ	Drop ratio (%)	Full part. ratio (%)	Acc_s (%)
0.0	0.00	100.0	97.68
0.2	9.72	75.1	97.65
0.4	29.28	43.2	97.68
0.6	39.73	34.3	97.43
0.8	46.27	25.9	97.11
1.0	49.30	24.0	97.04

Drop ratio: the average fraction of mini-batches in which a node misses the deadline τ (equivalently, the proportion of per-epoch training data failing to reach the server in time). Full part. ratio: fraction of mini-batches with no dropped node. Acc_s: student detection accuracy.

This performance preservation can be attributed to two structural properties of SFL-KD. First, the straggler drop mechanism is naturally compatible with minibatch SGD—a per-batch drop only perturbs the effective batch composition, a stochasticity that the learning process inherently tolerates. Second, the server-side KD smooths per-batch fluctuations by transferring soft teacher predictions to the student, acting as implicit regularization against intermittent communication noise. Overall, these complementary mechanisms enable SFL-KD to preserve its detection performance even under substantial communication dynamics. Note that dropped nodes

TABLE XI
COMPARISON OF THE PROPOSED FRAMEWORK WITH EXISTING DISTRIBUTED INTRUSION DETECTION APPROACHES

Study	Learning Framework	Local Efficiency (Train)	Server-independent Inference	Inference Opt.	Target Environment
Popoola et al. [14]	FL	×	✓	×	IoT-Edge
Almarshdi et al. [19]	FSL	✓	×	×	IoT
Jayasinghe et al. [29]	FL	×	✓	×	Beyond-5G
Maiga et al. [30]	FL	×	✓	×	5G Core
Singh et al. [31]	FL + KD	×	✓	×	Next-gen
Attanayaka et al. [32]	P2P FL	×	✓	×	Open RAN
Benzaïd et al. [33]	FL + CL	×	✓	×	Open RAN
Alalyan et al. [34]	P2P FL	×	✓	×	5G and Beyond
Pithani and Rout [35]	Clustered FL	×	✓	×	6G IoT-Edge
Park et al. [5]	SL	✓	×	×	5G
Ours (SFL-KD)	**SFL + KD**	✓	✓	✓	**5G, Open RAN, IoT**

TABLE XII
EMPIRICALLY MEASURED PER-ROUND TRAINING TIME ACROSS DISTRIBUTED LEARNING FRAMEWORKS

Phase	FL	SL	SFL-KD
Computation			
Local computation (constrained) [s]	13.2586	4.6266	0.5760
Server computation (unconstrained) [s]	–	10.5949	1.2798
Aggregation [s]	0.0084	–	0.0012
Total computation [s]	**13.2670**	**15.2215**	**1.8558**
Communication			
Data volume per round [MB]	3.19	1,003.84[a]	128.67[b]
Transfer time @ 1 Gbps [s]	0.0255	8.0307	1.0294
Transfer time @ 10 Gbps [s]	0.0026	0.8031	0.1029
Total per-round time cost (computation + communication)			
@ 1 Gbps [s]	13.2925	23.2522	**2.8852**
@ 10 Gbps [s]	13.2695	16.0246	**1.9588**

[a] In SL, each node sequentially transmits intermediate data to the server (125.48 MB × 8 nodes = 1,003.84 MB), where each node independently incurs its own processing time.
[b] In SFL-KD, all nodes transmit intermediate data in parallel (125.48 MB smashed data + 3.19 MB front-end parameters per node).

can reattempt in subsequent batches rather than being permanently excluded from training.

VI. DISCUSSION

To position the proposed framework relative to existing approaches, we first note that the baseline methods adopted in our experiments—namely FL, SL, and representative FL variants (FedProx, SCAFFOLD, and FedOpt)—were selected with the intent of providing an indirect yet systematic comparison against the broader landscape of state-of-the-art distributed intrusion detection systems. In particular, a substantial body of prior work in this domain adopts the standard FL framework as the underlying distributed learning paradigm while primarily varying the detection model architecture [14], [29], [30], [32], [33], [34], [35]. Since these studies share the same fundamental learning mechanism as the FL baseline evaluated in our experiments, their detection performance under comparable conditions can be reasonably approximated by our FL-based results. Similarly, the SL baseline in our experiments corresponds to the SL-based approach proposed by Park et al. [5], and the FL variant baselines (FedProx, SCAFFOLD, and FedOpt) reflect the class of optimization-enhanced FL methods adopted in recent studies [31]. From this perspective, the experimental results presented in Section V may offer relevant reference points for understanding the relative performance of the proposed framework in the context of existing distributed intrusion detection approaches. In addition to detection performance, we further compare the proposed framework with existing approaches from a functional perspective, focusing on operational efficiency on resource-constrained local nodes. Table XI summarizes a qualitative comparison of the proposed SFL-KD framework with representative distributed intrusion detection approaches targeting next-generation network environments. As shown in the table, the majority of existing studies adopt FL-based frameworks without incorporating mechanisms to reduce the computational burden on local nodes during either training or inference. While a few approaches employ model splitting [5], [19] to alleviate local training overhead, none of them address inference-phase efficiency through model compression or lightweight deployment. The proposed SFL-KD framework is, to the best of our knowledge, the first to jointly address both training- and inference-phase efficiency in distributed intrusion detection, by integrating model splitting with online KD within a unified framework.

In Section V, we individually analyzed the computational cost and communication overhead of the proposed framework

at each stage. However, a holistic assessment that jointly considers both factors in terms of end-to-end training overhead has not yet been addressed. To analyze the combined impact of computation and communication on overall training efficiency, we empirically measured the per-round training time for each distributed learning framework. In this experiment, we configured local nodes as resource-constrained environments (i.e., CPU-only processing), while the central server was assumed to be equipped with sufficient computational resources (i.e., multiple GPUs available). Table XII reports the resulting time breakdown under two representative network bandwidth conditions (1 and 10 Gb/s). Note that, in contrast to the stochastic transport scenarios examined in Section V-F4, this evaluation assumes idealized network conditions—stable bandwidth with no jitter or congestion—in order to enable a controlled comparison of computational and communication costs across frameworks. As shown in the table, the FL framework exhibits the highest local computation time (13.26 s) due to full-model training under constrained resources, while SL incurs a cumulative computation time of 15.22 s caused by its sequential relay structure across nodes. In contrast, SFL-KD achieves a total computation time of only 1.86 s by combining model splitting with parallel local updates. From a communication perspective, SL generates the largest per-round overhead (1003.84 MB) through sequential node-by-node transmission, whereas SFL-KD transmits 128.67 MB per node in parallel, keeping the effective communication time equivalent to a single node's transfer. When both factors are jointly considered, SFL-KD achieves the lowest per-round time cost across all evaluated conditions, requiring only 2.89 s at 1 Gb/s, while FL and SL require 13.29 and 23.25 s, respectively. These results confirm that FL is predominantly computation-bound, SL is communication-bound due to its sequential nature, and the proposed SFL-KD framework effectively mitigates both bottlenecks through parallel execution and model partitioning. In the experiments presented in this study, we configured eight local nodes based on the number of semantic classes in each dataset. While this configuration reflects a practical deployment scale, increasing the number of participating nodes would proportionally increase the aggregate volume of intermediate data transmitted to the central server. For instance, with the current per-node communication overhead of approximately 128.67 MB per round, scaling to 32 and 128 nodes would result in aggregate server ingress of approximately 3.8 and 16.1 GB per round, respectively. In such cases, even assuming a high-speed network interface of 10 Gb/s at the server side, the aggregate data transfer alone would require several seconds per round, and this overhead could become increasingly prohibitive as the number of participating nodes grows substantially. We note that this scalability concern is in line with a fundamental limitation of split-based distributed learning approaches, where intermediate activations must be exchanged between local nodes and the central server regardless of the specific framework design. To address this challenge in large-scale deployments, several promising directions can be explored, such as communication-efficient techniques including gradient compression and quantization, as well as hierarchical aggregation architectures that distribute the server-side workload across multiple intermediate servers.

From a privacy-preserving perspective, the proposed framework transmits intermediate activations (i.e., smashed data) from local nodes to the central server during training, which inherently involves the exchange of data-derived representations and may raise concerns regarding potential information leakage. To analyze the impact of privacy-preserving

TABLE XIII
PRIVACY–UTILITY TRADEOFF UNDER VARYING PRIVACY BUDGETS (ε) ON THE 5G-NIDD DATASET (NON-IID, $\alpha = 0.3$)

Privacy budget (ε)	Accuracy	Δ Accuracy
∞ (no DP)	0.9763	–
≥ 3	≥ 0.9712	≤ 0.51%p
2	0.9692	0.71%p
1	0.9524	2.39%p
0.5	0.9027	7.36%p

TABLE XIV
AGGREGATE PERFORMANCE METRICS ON THE OPEN RAN DATASET

Scenario	Algorithm	Accuracy	Macro F1	Weighted F1
–	Centralized	0.9260	0.9250	0.9256
IID	FL	0.9156	0.9141	0.9154
	SL	0.9171	0.9155	0.9166
	FedProx	0.9169	0.9153	0.9164
	SCAFFOLD	0.9214	0.9202	0.9213
	FedOpt	0.9170	0.9155	0.9165
	SFL-KD (ours)	0.9222	0.9208	0.9218
Non-IID	FL	0.9104	0.9080	0.9101
	SL	0.9062	0.9030	0.9059
	FedProx	0.8999	0.8956	0.8999
	SCAFFOLD	0.8988	0.8933	0.8972
	FedOpt	0.9046	0.9002	0.9039
	SFL-KD (ours)	0.9187	0.9167	0.9180

TABLE XV
CONFUSION MATRIX OF FL ON THE OPEN RAN DATASET

	Web	VoIP	IoT	Stream	DDoS	Hulk	PScan	Slow
Web	**40920**	601	1000	3678	230	114	39	1046
VoIP	710	**73292**	298	4	152	11	177	2110
IoT	539	174	**100308**	14	33	14	71	484
Stream	1580	38	154	**61568**	258	110	92	24
DDoS	511	192	101	311	**79925**	12191	4306	548
Hulk	463	152	168	160	11657	**99094**	860	248
PScan	220	419	318	30	3405	323	**86533**	982
Slow	1346	2181	807	22	356	43	794	**36519**

TABLE XVI
CONFUSION MATRIX OF SL ON THE OPEN RAN DATASET

	Web	VoIP	IoT	Stream	DDoS	Hulk	PScan	Slow
Web	**41654**	658	703	2946	176	102	46	1343
VoIP	680	**73306**	184	12	73	27	122	2350
IoT	1469	372	**98613**	35	44	31	94	979
Stream	2287	64	96	**60991**	197	97	71	21
DDoS	628	253	67	280	**77772**	13349	5125	611
Hulk	517	175	135	162	11084	**99451**	980	298
PScan	303	540	240	65	2548	389	**86978**	1167
Slow	1692	2271	386	3	294	62	685	**36675**

TABLE XVII
CONFUSION MATRIX OF FEDPROX ON THE OPEN RAN DATASET

	Web	VoIP	IoT	Stream	DDoS	Hulk	PScan	Slow
Web	**42015**	609	1116	2733	201	116	81	932
VoIP	820	**73157**	281	1	103	22	125	2118
IoT	1162	193	**99170**	30	23	17	106	687
Stream	3225	19	140	**59800**	265	146	88	24
DDoS	739	247	123	381	**79171**	12606	4225	564
Hulk	630	205	154	105	12426	**98290**	859	226
PScan	318	655	316	55	4116	355	**85042**	1615
Slow	2510	2712	909	9	463	60	597	**34820**

TABLE XVIII
CONFUSION MATRIX OF SCAFFOLD ON THE OPEN RAN DATASET

	Web	VoIP	IoT	Stream	DDoS	Hulk	PScan	Slow
Web	**40612**	805	1589	3342	186	223	87	784
VoIP	806	**72836**	604	14	152	48	176	2118
IoT	651	394	**100164**	23	22	23	71	289
Stream	2002	60	155	**61120**	189	186	97	15
DDoS	553	227	181	341	**73006**	16658	6665	454
Hulk	414	168	219	157	8649	**101762**	1233	200
PScan	219	474	537	49	1399	380	**88044**	1128
Slow	2366	2376	2879	12	393	80	733	**33229**

TABLE XIX
CONFUSION MATRIX OF FEDOPT ON THE OPEN RAN DATASET

	Web	VoIP	IoT	Stream	DDoS	Hulk	PScan	Slow
Web	**43176**	704	605	2472	60	103	29	654
VoIP	684	**74439**	72	3	18	31	53	1327
IoT	1471	264	**99017**	23	10	57	50	496
Stream	2876	49	103	**60320**	87	210	51	11
DDoS	939	414	85	337	**74747**	15782	4927	825
Hulk	616	218	121	99	8632	**102079**	871	259
PScan	301	867	250	28	2269	517	**85343**	2897
Slow	2471	3340	539	6	127	54	246	**35297**

TABLE XX
CONFUSION MATRIX OF SFL-KD (OURS) ON THE OPEN RAN DATASET

	Web	VoIP	IoT	Stream	DDoS	Hulk	PScan	Slow
Web	**41513**	984	889	2589	264	157	151	1081
VoIP	282	**74814**	151	12	98	21	196	1180
IoT	387	350	**100486**	24	21	21	163	185
Stream	1567	46	100	**61657**	194	105	136	19
DDoS	298	304	98	305	**79830**	11955	4890	405
Hulk	269	167	162	214	10256	**100509**	1052	173
PScan	92	365	197	56	1791	227	**88443**	1059
Slow	969	2988	900	14	331	53	678	**36135**

mechanisms on detection performance, we conducted additional experiments by applying a differentially private mechanism [46] to the smashed data. Specifically, we applied the Gaussian mechanism, where each smashed data vector was clipped to a bounded ℓ_2 sensitivity of 1, and calibrated Gaussian noise was added with $\delta = 10^{-5}$, following the standard Gaussian mechanism (see [47] for details on the application of DP in learning algorithms). The privacy budget ε was varied across a practical range from 0.5 to 10 to assess the impact on detection performance under the 5G-NIDD dataset with a non-IID data distribution ($\alpha = 0.3$). Table XIII summarizes the resulting privacy–utility tradeoff.

As shown in the table, under practical privacy budget settings ($\varepsilon \geq 2$), the proposed framework maintains detection accuracy above 0.9692, corresponding to less than 0.8 percentage points of degradation compared to the nonprivate baseline (0.9763). At $\varepsilon = 1$, the accuracy decreases to 0.9524, representing approximately 2.4 percentage points of degradation, which may still be acceptable in scenarios where privacy is prioritized. However, under a stronger privacy

TABLE XXI
AGGREGATE PERFORMANCE METRICS ON THE 5G-NIDD DATASET

Scenario	Algorithm	Accuracy	Macro F1	Weighted F1
–	Centralized	0.9830	0.9937	0.9830
IID	FL	0.9823	0.9930	0.9823
	SL	0.9825	0.9933	0.9825
	FedProx	0.9822	0.9927	0.9822
	SCAFFOLD	0.9826	0.9932	0.9825
	FedOpt	0.9824	0.9933	0.9825
	SFL-KD (ours)	0.9824	0.9933	0.9824
Skewed	FL	0.9419	0.8961	0.9408
	SL	0.9620	0.9417	0.9620
	FedProx	0.9006	0.6230	0.8931
	SCAFFOLD	0.9432	0.6336	0.9387
	FedOpt	0.9563	0.9316	0.9562
	SFL-KD (ours)	0.9770	0.9910	0.9770

TABLE XXII
CONFUSION MATRIX OF FL ON THE 5G-NIDD DATASET

	Benign	HTTP	ICMP	SYN-F	SYN-S	Slow	TCP-S	UDP-F	UDP-S
Benign	**91542**	805	44	26	12	371	46	2945	3
HTTP	53	**27753**	0	1	0	356	0	0	0
ICMP	0	0	**238**	0	0	0	0	0	0
SYN-F	0	1	0	**1657**	0	0	261	0	0
SYN-S	0	2	1	6	**3871**	0	14	0	0
Slow	5	840	0	0	0	**13618**	40	0	0
TCP-S	0	2	1	37	1	2	**4009**	0	1
UDP-F	6476	0	0	0	0	0	0	**84945**	0
UDP-S	0	1	0	1	993	1	0	782	**1415**

TABLE XXIII
CONFUSION MATRIX OF SL ON THE 5G-NIDD DATASET

	Benign	HTTP	ICMP	SYN-F	SYN-S	Slow	TCP-S	UDP-F	UDP-S
Benign	**89742**	579	94	33	42	325	64	4912	3
HTTP	62	**27605**	0	0	0	496	0	0	0
ICMP	0	0	**238**	0	0	0	0	0	0
SYN-F	0	0	0	**1702**	0	0	217	0	0
SYN-S	0	0	1	6	**3887**	0	0	0	0
Slow	0	374	0	0	0	**14086**	43	0	0
TCP-S	0	1	2	109	4	13	**3924**	0	0
UDP-F	1567	0	0	0	0	0	0	**89854**	0
UDP-S	3	0	1	1	7	0	0	280	**2901**

TABLE XXIV
CONFUSION MATRIX OF FEDPROX ON THE 5G-NIDD DATASET

	Benign	HTTP	ICMP	SYN-F	SYN-S	Slow	TCP-S	UDP-F	UDP-S
Benign	**95758**	0	0	3	2	5	24	0	2
HTTP	0	**24093**	0	0	4	4059	1	0	6
ICMP	0	0	**238**	0	0	0	0	0	0
SYN-F	0	0	0	**423**	544	313	639	0	0
SYN-S	5	0	0	1	**3819**	3	66	0	0
Slow	25	143	0	0	17	**14308**	10	0	0
TCP-S	1214	0	0	24	3	0	**2812**	0	0
UDP-F	13635	0	0	0	0	0	0	**77786**	0
UDP-S	3189	0	0	0	4	0	0	0	**0**

TABLE XXV
CONFUSION MATRIX OF SCAFFOLD ON THE 5G-NIDD DATASET

	Benign	HTTP	ICMP	SYN-F	SYN-S	Slow	TCP-S	UDP-F	UDP-S
Benign	**91061**	40	0	4	2	0	5	4682	0
HTTP	3	**27826**	0	0	0	333	1	0	0
ICMP	0	0	**0**	0	0	238	0	0	0
SYN-F	1	0	0	**1627**	272	17	2	0	0
SYN-S	0	0	0	13	**3861**	0	20	0	0
Slow	10	368	0	5	14110	**10**	0	0	0
TCP-S	1	0	0	10	3083	2	**957**	0	0
UDP-F	1502	0	0	0	0	0	0	**89919**	0
UDP-S	1	0	0	3	3045	0	144	0	**0**

TABLE XXVI
CONFUSION MATRIX OF FEDOPT ON THE 5G-NIDD DATASET

	Benign	HTTP	ICMP	SYN-F	SYN-S	Slow	TCP-S	UDP-F	UDP-S
Benign	**89019**	0	0	0	0	2	0	6773	0
HTTP	7	**26404**	0	38	0	1714	0	0	0
ICMP	0	0	**238**	0	0	0	0	0	0
SYN-F	1	1	0	**1366**	262	1	288	0	0
SYN-S	0	0	4	4	**3886**	0	0	0	0
Slow	37	886	0	168	0	**13369**	43	0	0
TCP-S	36	0	5	0	2	0	**4010**	0	0
UDP-F	124	0	0	0	0	0	0	**91297**	0
UDP-S	0	0	8	0	232	0	0	0	**2953**

TABLE XXVII
CONFUSION MATRIX OF SFL-KD (OURS) ON THE 5G-NIDD DATASET

	Benign	HTTP	ICMP	SYN-F	SYN-S	Slow	TCP-S	UDP-F	UDP-S
Benign	**91756**	3	0	1	0	0	2	4032	0
HTTP	1	**28028**	0	0	0	134	0	0	0
ICMP	0	0	**238**	0	0	0	0	0	0
SYN-F	1	0	1	**1917**	0	0	0	0	0
SYN-S	0	0	0	5	**3878**	0	3	0	8
Slow	0	92	0	0	0	**14411**	0	0	0
TCP-S	13	0	0	0	0	0	**4036**	0	4
UDP-F	1285	0	0	0	0	0	0	**90136**	0
UDP-S	0	0	0	0	2	0	3	0	**3188**

guarantee ($\varepsilon = 0.5$), a more substantial performance drop to 0.9027 is observed, indicating that aggressive noise injection significantly disrupts the informative content of the intermediate representations. We attribute the relatively moderate performance degradation at practical privacy levels primarily to the compact dimensionality of the smashed data at cut layer 2 (128 features), which limits the information exposed per transmission and allows the model to retain sufficient discriminative capacity. While these preliminary results suggest that a practical privacy–utility tradeoff is achievable within the proposed framework, the design of more sophisticated privacy-preserving mechanisms—such as adaptive noise calibration and structured masking techniques—remains an important direction for future research (see [48] for a detailed analysis of privacy tradeoffs in distributed learning settings).

VII. CONCLUSION

This article presents a split-federated learning framework with a shadow-augmented KD strategy for distributed AI-based intrusion detection in resource-constrained environments. The proposed framework addresses the limitations of existing FL- and SL-based approaches by combining the model partitioning of SL with the parallel aggregation of FL and by embedding an online distillation process to optimize a lightweight student model during training. This integration enables each local node to operate independently for real-time threat detection, substantially reducing computational overhead while maintaining competitive detection accuracy. Comprehensive evaluations on the 5G-NIDD and Open RAN datasets demonstrate that SFL-KD achieves performance comparable to centralized training while preserving efficiency across diverse data distribution scenarios. Moreover, the results highlight the capability of the proposed framework to maintain stable convergence under heterogeneous conditions, effectively mitigating the instability issues inherent in FL and the inefficiency of SL. These findings substantiate the practicality of SFL-KD for enhancing network security in large-scale distributed environments characterized by heterogeneity and resource constraints.

We expect that the proposed framework establishes a fundamental distributed AI-NIDS paradigm that jointly considers efficiency in both training and inference and, thus, offers broad avenues for extension. In essence, future work may explore diverse model configurations built upon our framework, including adaptive model compression, dynamic cut-layer selection, and integration with hierarchical distributed learning. Building on this foundation, several promising directions can be pursued: 1) designing advanced privacy-preserving mechanisms—such as adaptive noise calibration and structured masking—along with robustness-oriented aggregation strategies to improve resilience against adversarial attacks; 2) incorporating secure model exchange mechanisms (e.g., secure multiparty computation); and 3) adopting communication-efficient techniques such as gradient compression and quantization to reduce the data exchange overhead in large-scale deployments.

APPENDIX A
ADDITIONAL EVALUATION METRICS AND CONFUSION MATRICES ON THE OPEN RAN DATASET

This appendix provides aggregate performance metrics (Table XIV) and per-class confusion matrices (Tables XV–XX) for each distributed learning framework evaluated on the Open RAN dataset under the NonIID scenario.

APPENDIX B
ADDITIONAL EVALUATION METRICS AND CONFUSION MATRICES ON THE 5G-NIDD DATASET

This appendix provides aggregate performance metrics (Table XXI) and per-class confusion matrices (Tables XXII–XXVII) for each distributed learning framework evaluated on the 5G-NIDD dataset under the skewed scenario.

REFERENCES

[1] S. Dang, O. Amin, B. Shihada, and M.-S. Alouini, "What should 6G be?" *Nature Electron.*, vol. 3, no. 1, pp. 20–29, Jan. 2020.
[2] C.-X. Wang et al., "On the road to 6G: Visions, requirements, key technologies, and testbeds," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 905–974, 2nd Quart. 2023.
[3] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016, *arXiv:1602.05629*.

[4] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, Aug. 2018.

[5] C. Park, K. Park, J. Song, and J. Kim, "Distributed learning-based intrusion detection in 5G and beyond networks," in *Proc. Joint Eur. Conf. Net. Commun. 6G Summit*, Apr. 2023, pp. 490–495.

[6] S. Samarakoon, "5G-NIDD: A comprehensive network intrusion detection dataset generated over 5G wireless network," *IEEE Dataport*, doi: 10.21227/xtep-hv36.

[7] B. M. Xavier, M. Dzaferagic, M. Martinello, and M. Ruffini, "Performance measurement dataset for open RAN with user mobility and security threats," *Comput. Netw.*, vol. 253, Nov. 2024, Art. no. 110710.

[8] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "DÏoT: A federated self-learning anomaly detection system for IoT," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 756–767.

[9] S. A. Rahman, H. Tout, C. Talhi, and A. Mourad, "Internet of Things intrusion detection: Centralized, on-device, or federated learning?" *IEEE Netw.*, vol. 34, no. 6, pp. 310–317, Nov. 2020.

[10] Q. Qin, K. Poularakis, K. K. Leung, and L. Tassiulas, "Line-speed and scalable intrusion detection at the network edge via federated learning," in *Proc. IFIP Netw. Conf. (Netw.)*, Jun. 2020, pp. 352–360.

[11] T. T. Huong et al., "LocKedge: Low-complexity cyberattack detection in IoT edge computing," *IEEE Access*, vol. 9, pp. 29696–29710, 2021.

[12] J. Zhang, P. Yu, L. Qi, S. Liu, H. Zhang, and J. Zhang, "FLDDoS: DDoS attack detection model based on federated learning," in *Proc. IEEE 20th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Oct. 2021, pp. 635–642.

[13] N. Hamdi, "Federated learning-based intrusion detection system for Internet of Things," *Int. J. Inf. Secur.*, vol. 22, no. 6, 2023, pp. 1937–1948.

[14] S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, and O. Jogunola, "Federated deep learning for zero-day botnet attack detection in IoT-edge devices," *IEEE Internet Things J.*, vol. 9, no. 5, pp. 3930–3944, Mar. 2022.

[15] R. B. Chaabene, D. Ameyed, F. Jaafar, A. Roger, A. Esma, and M. Cheriet, "A privacy-preserving federated learning for IoT intrusion detection system," in *Proc. 9th Int. Conf. Control, Decis. Inf. Technol. (CoDIT)*, Jul. 2023, pp. 351–356.

[16] X. Wang, Y. Wang, Z. Javaheri, L. Almutairi, N. Moghadamnejad, and O. S. Younes, "Federated deep learning for anomaly detection in the Internet of Things," *Comput. Electr. Eng.*, vol. 108, May 2023, Art. no. 108651.

[17] S. Yılmaz, S. Sen, and E. Aydogan, "Exploring and enhancing placement of IDS in RPL: A federated learning-based approach," *IEEE Internet Things J.*, vol. 12, no. 13, pp. 22945–22961, Jul. 2025.

[18] J. Ma and W. Su, "Collaborative DDoS defense for SDN-based AIoT with autoencoder-enhanced federated learning," *Inf. Fusion*, vol. 117, May 2025, Art. no. 102820.

[19] R. Almarshdi, E. Fadel, N. Alowidi, and L. Nassef, "Distributed federated split learning based intrusion detection system," *Intell. Autom. Soft Comput.*, vol. 39, no. 5, pp. 949–983, 2024.

[20] M. Itani, H. Basheer, and F. Eddine, "Attack detection in IoT-based healthcare networks using hybrid federated learning," in *Proc. Int. Conf. Smart Appl., Commun. Netw. (SmartNets)*, Jul. 2023, pp. 1–7.

[21] Y. Otoum, V. Chamola, and A. Nayak, "Federated and transfer learning-empowered intrusion detection for IoT applications," *IEEE Internet Things Mag.*, vol. 5, no. 3, pp. 50–54, Sep. 2022.

[22] A. A. Alharbi, "Federated transfer learning for attack detection for Internet of Medical Things," *Int. J. Inf. Secur.*, vol. 23, no. 1, pp. 81–100, Feb. 2024.

[23] M. Fahim-Ul-Islam, A. Chakrabarty, M. G. R. Alam, and S. S. B. Maidin, "A resource-efficient federated learning framework for intrusion detection in IoMT networks," *IEEE Trans. Consum. Electron.*, vol. 71, no. 2, pp. 4508–4521, May 2025.

[24] T. V. Khoa et al., "Collaborative learning model for cyberattack detection systems in IoT Industry 4.0," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, May 2020, pp. 1–6.

[25] J. Li, L. Lyu, X. Liu, X. Zhang, and X. Lyu, "FLEAM: A federated learning empowered architecture to mitigate DDoS in industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 18, no. 6, pp. 4059–4068, Jun. 2022.

[26] P. Ruzafa-Alcázar et al., "Intrusion detection based on privacy-preserving federated learning for the industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1145–1154, Feb. 2023.

[27] B. Li, Y. Wu, J. Song, R. Lu, T. Li, and L. Zhao, "DeepFed: Federated deep learning for intrusion detection in industrial cyber–physical systems," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5615–5624, Aug. 2021.

[28] A. Makkar, T. W. Kim, A. K. Singh, J. Kang, and J. H. Park, "SecureIIoT environment: Federated learning empowered approach for securing IIoT from data breach," *IEEE Trans. Ind. Informat.*, vol. 18, no. 9, pp. 6406–6414, Sep. 2022.

[29] S. Jayasinghe, Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "Federated learning based anomaly detection as an enabler for securing network and service management automation in beyond 5G networks," in *Proc. Joint Eur. Conf. Netw. Commun. 6G Summit (EuCNC/6G Summit)*, Jun. 2022, pp. 345–350.

[30] A.-A. Maiga, E. Ataro, and S. Githinji, "XGBoost and deep learning based-federated learning for DDoS attack detection in 5G core network VNFs," in *Proc. 6th Int. Conf. Comput. Commun. Internet (ICCCI)*, Jun. 2024, pp. 128–133.

[31] G. Singh, K. Sood, P. Rajalakshmi, D. D. N. Nguyen, and Y. Xiang, "Evaluating federated learning-based intrusion detection scheme for next generation networks," *IEEE Trans. Netw. Service Manage.*, vol. 21, no. 4, pp. 4816–4829, Aug. 2024.

[32] D. Attanayaka, P. Porambage, M. Liyanage, and M. Ylianttila, "Peer-to-peer federated learning based anomaly detection for open radio access networks," in *Proc. IEEE Int. Conf. Commun.*, May 2023, pp. 5464–5470.

[33] C. Benzaïd, F. M. Hossain, T. Taleb, P. M. Gómez, and M. Dieudonne, "A federated continual learning framework for sustainable network anomaly detection in O-RAN," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2024, pp. 1–6.

[34] F. Alalyan, B. Bousalem, W. Jaafar, and R. Langar, "Secure peer-to-peer federated learning for efficient cyberattacks detection in 5G and beyond networks," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2024, pp. 1752–1757.

[35] A. Pithani and R. R. Rout, "CFL-ATELM: An approach to detect botnet traffic by analyzing non-IID and imbalanced data in IoT-edge based 6G networks," *Cluster Comput.*, vol. 28, no. 3, p. 198, Jun. 2025.

[36] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Mach. Learn. Syst.*, 2020, pp. 429–450.

[37] G. Damaskinos, E. M. El-Mhamdi, R. Guerraoui, A. Guirguis, and S. Rouault, "AGGREGATHOR: Byzantine machine learning via robust gradient aggregation," in *Proc. Mach. Learn. Syst.*, 2019, pp. 81–106.

[38] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "SplitFed: When federated learning meets split learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, 2022, pp. 8485–8493.

[39] S. Nair, M. Lin, P. Ju, A. Talebi, E. S. Bentley, and J. Liu, "FSL-SAGE: Accelerating federated split learning via smashed activation gradient estimation," in *Proc. Int. Conf. Mach. Learn.*, 2025, pp. 45507–45528.

[40] Z. Lin et al., "Hierarchical split federated learning: Convergence analysis and system optimization," *IEEE Trans. Mobile Comput.*, vol. 24, no. 10, pp. 9352–9367, Oct. 2025.

[41] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[42] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Proc. 37th Int. Conf. Mach. Learn.*, vol. 119, Jul. 2020, pp. 5132–5143.

[43] S. J. Reddi et al., "Adaptive federated optimization," in *Proc. Int. Conf. Learn. Represent.*, 2021.

[44] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, "CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment," *Sensors*, vol. 23, no. 13, p. 5941, Jun. 2023.

[45] F. Ullah, M. Nadeem, M. Abrar, F. Amin, A. Salam, and S. Khan, "Enhancing brain tumor segmentation accuracy through scalable federated learning with advanced data privacy and security measures," *Mathematics*, vol. 11, no. 19, p. 4189, Oct. 2023.

[46] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, nos. 3–4, pp. 211–487, Aug. 2014.

[47] M. Abadi et al., "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 308–318.

[48] A. Salam, M. Abrar, F. Ullah, I. A. Khan, F. Amin, and G. S. Choi, "Efficient data collaboration using multi-party privacy preserving machine learning framework," *IEEE Access*, vol. 11, pp. 138151–138164, 2023.

Cheolhee Park received the B.S. degree from the Department of Applied Mathematics, Kongju National University, Gongju, South Korea, in 2014, and the M.S. and Ph.D. degrees from the Department of Mathematics, Kongju National University, in 2017 and 2021, respectively.

He joined the Electronics and Telecommunications Research Institute (ETRI), Daejeon, Republic of Korea, in 2021, where he is currently a Researcher. His research interests include data privacy, cloud security, deep learning (DL), artificial intelligence (AI) security, network security, and 5G/6G security.

Kyungmin Park received the B.S., M.S., and Ph.D. degrees in computer engineering from Chungnam National University, Daejeon, Korea, in 2010, 2013, and 2019, respectively.

He joined the Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea, in 2017, where he is currently in the Intelligent Network Research Section. His research interests include 5G/6G cellular networks and intelligent network architecture.

Jihyeon Song received the M.S. and Ph.D. degrees in information security engineering from the University of Science and Technology (UST), Daejeon, South Korea, in 2020 and 2026, respectively.

She was a Postdoctoral Researcher with the Department of Computer and Information Security, Sejong University, Seoul, South Korea. Her research interests include network security, malware analysis, and artificial intelligence (AI)-based security.

Soohyung Kim received the B.S. and M.S. degrees in computer science from Yonsei University, Seoul, Korea, in 1996 and 1998, respectively, and the Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2016.

He joined the Electronics and Telecommunications Research Institute (ETRI), Daejeon, in 2000, where he is currently a Principal Researcher with the Cyber Security Research Division. His research interests include identity management, biometric authentication, data privacy, 5G/6G security, and artificial intelligence (AI) security.

Jonghyun Kim received the Ph.D. degree in computer science from The University of Oklahoma, Norman, OK, USA, in 2005.

He was a Researcher with Samsung Electronics, Suwon, South Korea, from 1995 to 1997. He was a Principal Researcher with the Electronics Telecommunications Research Institute, Daejeon, South Korea, from 2005 to 2024. He is currently a Professor with the Department of Computer and Information Security, Sejong University, Seoul, South Korea. He was also involved in standardization activities as the Vice Chair of WP1 and a rapporteur of Q.4 (cybersecurity) with the ITU-T Study Group 17. His research interests include information security, cybersecurity, cloud security, artificial intelligence (AI)-based malware detection, and 5G/6G security.