This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3309633

RingSFL: An Adaptive Split Federated Learning Towards Taming Client Heterogeneity

Jinglong Shen, *Student Member, IEEE,* Nan Cheng, *Member, IEEE,*
Xiucheng Wang, *Student Member, IEEE,* Feng Lyu, *Member, IEEE,* Wenchao Xu, *Member, IEEE,*
Zhi Liu, *Member, IEEE,* Khalid Aldubaikhy, *Member, IEEE,* and Xuemin (Sherman) Shen, *Fellow, IEEE*

Abstract—Federated learning (FL) has gained increasing attention due to its ability to collaboratively train while protecting client data privacy. However, vanilla FL cannot adapt to client heterogeneity, leading to a degradation in training efficiency due to stragglers, and is still vulnerable to privacy leakage. To address these issues, this paper proposes RingSFL, a novel distributed learning scheme that integrates FL with a model split mechanism to adapt to client heterogeneity while maintaining data privacy. In RingSFL, all clients form a ring topology. For each client, instead of training the model locally, the model is split and trained among all clients along the ring through a pre-defined direction. By properly setting the propagation lengths of heterogeneous clients, the straggler effect is mitigated, and the training efficiency of the system is significantly enhanced. Additionally, since the local models are blended, it is less likely for an eavesdropper to obtain the complete model and recover the raw data, thus improving data privacy. The experimental results on both simulation and prototype systems show that RingSFL can achieve better convergence performance than benchmark methods on independently identically distributed (IID) and non-IID datasets, while effectively preventing eavesdroppers from recovering training data.

Index Terms—Federated Learning, Split Learning, Heterogeneity, Ring Topology, Straggler Effect.

◆

1 INTRODUCTION

Recently, machine learning (ML) techniques have been widely applied to various domains such as computer vision [1], natural language processing [2], and speech recognition [3] due to their remarkable representation and learning capabilities [4]. Typically, ML necessitates a large amount of data and computational resources to train a model with satisfactory generalization performance. Consequently, centralized learning has been extensively adopted, where a central server owns all the data and trains a model with abundant computing resources. Nevertheless, in many applications, training data is generated by users, and uploading such raw data to the cloud server may compromise user privacy. Moreover, as the computational and storage capabilities of the user devices increase exponentially, it becomes feasible to leverage the local resources for training tasks. In 2016, Google proposed federated learning (FL) [5], which has been gaining increasing attention. With FL, the ML models are trained over user devices while keeping data localized. Rather than raw data, local updated parameters are uploaded to the server for aggregation, which prevents data leakage.

Despite the potential of FL in edge networks, there remain numerous challenges [6], [7]. One of the most significant is the high heterogeneity among the clients involved in training [8]. This heterogeneity can manifest in terms of computational capability and battery level, both of which can have a significant impact on the efficiency of the FL system [9]. For instance, the different training times among clients due to uneven computational capabilities can lead to the *straggler effect*, where stragglers can bottleneck the efficiency of the FL system. Additionally, clients may have different battery levels, and the training process can drain the batteries of clients with lower levels, causing them to quit the FL system and making their training data inaccessible.

Another primary concern of FL is user privacy since sensitive information can still be revealed from model parameters/gradients by a third-party entity or the server. In [10], an optimization-based approach is proposed to recover user data from a single sample of gradients. This method involves the generation of random dummy inputs and labels locally, followed by the minimization of the distance between the dummy and actual gradients to recover the user data. Subsequently, [11] extends this approach by using a similar optimization method to achieve recovery of user data from the batch's average gradients. These works highlight the need for further improvements in the data privacy of FL.

In this paper, we propose a novel FL scheme, **RingSFL**, which integrates **FL** with a model **S**plit mechanism to address the above issues of FL. Specifically, clients form a ring topology where adjacent clients can communicate

---

- *J. Shen, N. Cheng, and X. Wang are with the School of Telecommunications Engineering, Xidian University, Xi'an, China. E-mail: jlshen@stu.xidian.edu.cn, dr.nan.cheng@ieee.org, xcwang_1@stu.xidian.edu.cn*
- *F. Lyu is with the School of Computer Science and Engineering, Central South University, Changsha, China. E-mail: fenglyu@csu.edu.cn*
- *W. Xu is with the Department of Computing, Hong Kong Polytechnic University, Hongkong, China. E-mail: wenchao.xu@polyu.edu.hk*
- *Z. Liu is with the Department of Computer and Network Engineering, The University of Electro-Communications, Chofugaoka, Japan. E-mail: liu@ieee.org*
- *K. Aldubaikhy is with the Department of Electrical Engineering, Qassim University, Buraydah, Saudi Arabia. E-mail: khalid@qec.edu.sa*
- *X. Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada. E-mail: sshen@uwaterloo.ca*

(Corresponding author: Nan Cheng.)

with each other through wireless links. The server assigns a *propagation length* (the number of neural layers to be processed during each forward and backward propagation) to each client according to the system heterogeneity, and the forward and backward propagation of each client starts from itself and traverses the ring topology under the constraint of propagation length to complete the training. After a round of training, each client has an updated model with different layers trained by data from different clients, i.e., a *blended model*, and the models are uploaded to the server for aggregation. RingSFL not only preserves the capability of FL to utilize the distributed computational power but also enhances data privacy since an eavesdropper can hardly recover the data from the blended model. Moreover, RingSFL can better adapt to the system heterogeneity by allocating computational loads according to the characteristics (such as computational capability or battery power) of different clients, which can significantly mitigate the straggler effect and improve the training efficiency of the system. In summary, the main contributions of this paper are as follows.

- *Novel adaptive and privacy-preserving FL scheme*: A novel FL scheme, termed RingSFL, is proposed. This scheme adaptively distributes the overall training load through a model split mechanism, thereby migrating the computational load from weak computational power clients to strong computational power clients, thus alleviating the straggler effect of vanilla FL, improving the computational efficiency of the system, and reducing the training time. Additionally, RingSFL not only does not transmit user data, but also does not transmit any complete model updates. All model updates uploaded to the server are blended through the model split mechanism, thus achieving enhanced privacy preservation.
- *Improved Model Performance*: The splitting of the model at the logical level in RingSFL, through the assignment of propagation lengths, has been observed to lead to the emergence of **overlapping layers** that are trained by multiple clients in parallel. These overlapping layers have a higher frequency of gradient aggregation, resulting in more reliable gradients. Consequently, a higher model accuracy can be attained by utilizing an adaptive learning rate to augment the step size of the gradients of the overlapping layers.
- *Simulation and Prototype System*: The efficacy and performance of the proposed RingSFL scheme were evaluated using real datasets in both a simulation environment and a prototype system containing two PC nodes and three Raspberry Pi nodes. The experimental results validate the effectiveness of the proposed RingSFL scheme.

The remainder of the paper is organized as follows. Section 2 provides the background and related work of the paper. Section 3 describes the proposed RingSFL, including the design of the training process, model split scheme, model aggregation scheme, improvement based on overlapping layers, and the discussion on the privacy enhancement. In section 4, extensive experimental results are given to evaluate RingSFL, followed by limitations and future works discussed in section 5. Finally, concluding remarks are given in section 6.

2 BACKGROUND AND RELATED WORK

2.1 Federated Learning

FL [5] aims to train a high-quality centralized model by leveraging the data distributed across numerous devices and addressing the issue of data silos. The process begins with the server initializing a global model and sending it to the participating clients. Each client trains the model using its own local dataset and sends the updated model parameters to the server. The server aggregates the received model parameters to create the new global model, which is then distributed back to the participating clients. The process is iteratively repeated until the global model converges.

Client Heterogeneity in FL: For client heterogeneity, an effective solution is to design a novel user scheduling mechanism. By scheduling high-quality clients (e.g., with sufficient computing resources, communication resources, stable communication connections, etc.) to participate in training, the straggler effect caused by low-quality clients is mitigated. The commonly used scheduling schemes include: reinforcement learning [12], long-term perspective scheduling [13], joint optimization [14], etc. But there may be a lot of valuable data on straggler, and not scheduling them would result in the absence of these data. Another effective approach is to design novel resource allocation strategies to mitigate client heterogeneity, such as allocating additional resources to low-quality clients [15], [16]. Nevertheless, most of the work focuses on allocating communication resources or computational resources on the server, and still lacks a method to efficiently allocate computational resources on the client side.

Privacy Protection in FL: Several research works have made essential efforts to further improve the privacy of FL. One popular approach is differential privacy, which is known for its low-complexity advantage and better privacy is achieved by adding white noise to the model [17], [18], [19], [20]. However, the addition of white noise usually results in degraded model performance, necessitating further investigation into the trade-off between model performance and privacy. Cryptographic mechanisms, such as secure aggregation studied in [21], [22], that use an encryption mechanism to prevent any party, including the server, from accessing each client's uploaded model, are becoming more mainstream. Other approaches are generating samples instead of using actual samples for training [23] and designing blockchain-based FL systems to enhance the FL system's security [24], [25]. However, cryptographic mechanisms generally impose additional computational burdens, which may not be tolerated by low-resource clients and lead to a more severe straggler effect.

2.1.1 Benefits and Limitations

FL has been demonstrated to achieve accuracy similar to centralized learning on IID datasets. Additionally, FL offers the advantage of parallel training, resulting in relatively low training time consumption. Nevertheless, FL is still susceptible to client heterogeneity, leading to the straggler effect. Existing solutions may lead to the absence of valuable data [12], [13], [14] and lack a way to efficiently allocate computational resources on the client side [15], [16]. Motivated by this, RingSFL achieves the allocation of training load based on client capabilities, enabling stragglers to participate in training with a smaller training load, thus avoiding the loss of valuable data from stragglers. Moreover, Fl still has the risk of privacy leakage, and existing solutions may sacrifice global model accuracy [17], [18], [19], [20] or lead to intolerable computational burden [21], [22], [23], [24], [25]. Conversely, RingSFL enhances privacy while obtaining better global model accuracy without additional computation. It is worth noting that RingSFL is also compatible with mechanisms such as differential privacy and encryption.

2.2 Split Learning

The method of Split Learning (SL) has been introduced in [26] to improve the efficiency and security of distributed learning systems. In SL, the model is split into two parts and trained by a client and a server. In the training phase, the client inputs the local training data into the local model to perform forward propagation until the split layer's output is obtained. Afterward, the client transmits the split layer's output and associated labels to the server. The server feeds the client's output into the local model to proceed with the forward propagation and predict the model output. Lastly, the server estimates the loss value by using the model output and labels, then starts the back propagation process.

In the vanilla SL architecture described above [26], the server cannot directly access the raw data of clients, and the complete model parameters are not transmitted to the server. The only information exchanged between clients and the server is the output of the split layer from clients to the server, and the gradient of the split layer from the server to clients. However, this vanilla SL approach necessitates the transmission of labels, thus risking data leakage. To address this issue, a U-shaped structure is proposed in [27], wherein the deepest and shallowest layers are kept in the client and the middle layers are kept in the server, allowing the labels to remain in the client for training and thus improving system security. Further, random perturbation techniques are proposed in [28] to prevent label information leakage. Additionally, [29] proposes broadcasting an average gradient at the split layer during back propagation, thus enabling scalable parallel SL.

2.2.1 Benefits and Limitations

Vanilla SL reduces the client training load by splitting the model, rendering it more appropriate for edge network deployment in comparison to vanilla FL. Nevertheless, vanilla SL's inability to train in parallel leads to linearly increasing training latency with the number of clients and unsatisfactory convergence on Non-IID datasets [26], [27]. While some studies [28], [29] have tried to enable parallel training in SL, it still faces convergence challenges on Non-IID datasets. Additionally, we highlight that although SL reduces the client's training burden, each client still has an equal training workload and cannot distribute it according to its capability. Consequently, we introduce RingSFL, which allocates training load based on the client's capabilities, thereby improving model accuracy significantly on the Non-IID dataset.

2.3 Integration of Federated Learning and Split Learning

Recently, a novel approach to FL has been proposed, which seeks to integrate FL with SL in order to leverage their respective strengths and create a new architecture for distributed learning that is suitable for edge network environments. SplitFed, as described in [30], is a pioneering and successful example of this approach. The framework necessitates the inclusion of two servers, a main server for splitting training and a fed server for aggregating models. SplitFed splits the neural network between the clients and the main server. The clients then use SL with the main server to update their local model parameters. Subsequently, the clients transmit the updated local model parameters to the fed server for aggregation, resulting in a new global model and the commencement of the next round of training.

In [31], [32], the authors introduced local loss signals into SplitFed, allowing the client's subnetwork to train directly using local losses without receiving loss gradients from the server, reducing the communication cost of the system. In [33], the authors integrate FL with SL in an edge unmanned aerial vehicle (UAV) network. During training, some of the drones in the system perform SL with the base station, while the remaining drones perform FL, and finally all the model parameters are aggregated to get an updated global model. By controlling the training method (FL or SL) of the UAVs, a stronger adaptation to the edge network environment is achieved. FedSL is proposed in [34] for distributed training of recurrent neural networks (RNNs). The authors split an RNN into multiple sub-networks and distribute them to different clients for training. During local training, the sub-networks on different clients communicate with each other to capture potential dependencies between data on different clients. Finally, all clients send their sub-networks to the server for aggregation. In [35], each client has its corresponding edge server, and the clients perform split learning with their edge servers. After local training, the model parameters on the edge server are sent to the parameter server for aggregation. In the field of pre-training large models, integrating FL with SL can also play an active role. FedBert, as proposed in [36], successfully deploys the pre-training task of large models at the edge of the network by splitting the large model into many smaller models. Different from previous discussed works that integrate SL with horizontal FL, PyVertical [37] effectively combines SL with vertical FL, enabling the training of neural networks on vertically split data features between multiple clients while keeping the original data on the owner's device.

2.3.1 Benefits and Limitations

Integrating FL with SL retains their respective benefits, such as reduced client training loads in comparison to FL and enhanced parallel training capability and convergence performance than SL. However, these existing approaches assign the same training load to all clients, resulting in ineffective use of computational resources. Besides, they frequently depend on multiple servers that are challenging to implement in edge network conditions [30], [31], [32], [33], [34], [35]. In this paper, we effectively implement the training load allocation based on client capabilities by

TABLE 1
Notations Used in This Paper.

Notation	Explanation
$\mathcal{U}$	Client set containing clients participating in training.
u_i	Client indexed by i.
N	Number of clients participating in training.
$\mathcal{D}_i$	Local dataset of u_i.
D_i	Dataset size of $\mathcal{D}_i$.
C_i	Computational power of u_i.
$\mathcal{W}^t$	Global model at communication round t.
$\boldsymbol{x}_i$	Mini-batch sampled from $\mathcal{D}_i$.
y_i	Corresponding label of $\boldsymbol{x}_i$.
L_i	Propagation length.
a_i	Aggregation weight.
$\tilde{\boldsymbol{x}}_i$	Feature map of $\boldsymbol{x}_i$.
$loss\_fn$	Loss function.
$\boldsymbol{g}_i^t$	Gradient of $\mathcal{W}_i^t$.
p_i	Model split ratio.
M	Computation volume to complete mini-batch training.
$\tilde{\mathcal{W}}_{i,(j,k)}^t$	Back propagation process of $\mathcal{W}_{i,(j,k)}^t$.
$\mathcal{W}_{i,(j,k)}^t$	Layer j to k of u_i's local model at communication round t.
$\mathcal{U}_{i,(j)}$	The set of clients who propagate through the j-th layer of u_i's local model.
e_i'	The probability that the communication link between u_i and u_{i-1} is eavesdropped.
e_i	The probability that the communication link between u_i and server is eavesdropped.

controlling the propagation length of clients. Differing from existing methods, our approach utilizes only one server for training, promising wider applicability in edge network scenarios.

3 PROPOSED RINGSFL SCHEME

3.1 Overview

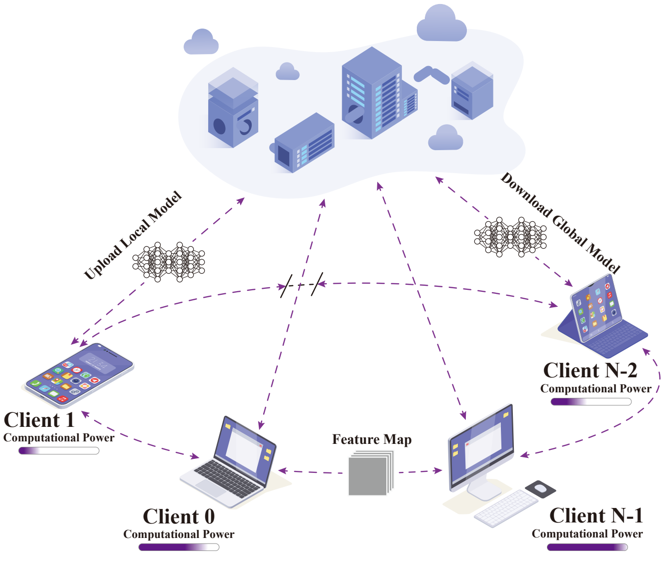

Fig. 1. The architecture of RingSFL with N clients.

The structure of the proposed RingSFL system is depicted in Fig. 1. This system consists of a server for model aggregation and a client set $\mathcal{U} = \{u_0, u_1, \cdots, u_{N-1}\}$ of N clients for cooperative training. Each client u_i has a training dataset $\mathcal{D}_i$ of size D_i, and the computational power of u_i is denoted by C_i. The clients form a ring topology, where adjacent clients can communicate with each other through direct communication technologies such as device-to-device (D2D) communication [38]. The clients can also communicate with the server for model downloading and uploading as in FL. It is worth noting that the construction of the ring topology will have a significant effect on the performance. Nevertheless, in this paper, we focus on the fundamental properties of RingSFL and then arbitrarily set the ring topology, leaving the investigation on ring topology construction as future work. The notations used in this paper are summarized in Table 1.

As illustrated in Fig. 2, during each communication round, multiple forward and backward propagation of a client is conducted cooperatively by all clients along the ring topology. To this end, the server assigns a *propagation length* to each client based on its capabilities, which is used to determine the number of neural layers each client needs to process during each forward and backward propagation. Subsequently, the forward propagation traverses the ring topology, and each client is responsible for propagating the number of neural layers specified by the propagation length over its local model. Similarly, the backward propagation is performed by each client to compute the gradients of the corresponding layers. As the example in Fig. 2, if there are three clients, denoted by u_0, u_1, and u_2, and the propagation lengths are set to 2:1:3, respectively. Then, the model of u_0 is trained traversing $u_0 \to u_1 \to u_2$ for forward propagation and $u_2 \to u_1 \to u_0$ for backward propagation (blue arrows). Similarly, the model of u_1 is trained traversing $u_1 \to u_2 \to u_0$ for forward propagation and $u_0 \to u_2 \to u_1$ for backward propagation (green arrows), and the model of u_2 is trained traversing $u_2 \to u_0 \to u_1$ for forward propagation and $u_1 \to u_0 \to u_2$ for backward propagation (red arrows). It is worth noting that the training process of different clients are independent and are conducted simultaneously. Finally, the trained local models are transmitted to the server for aggregation. This process is repeated until the model converges, as detailed in Algorithm 1.

3.2 RingSFL Training Process

Without loss of generality, we present the detailed training process of RingSFL for an arbitrary number of clients in this subsection. The local model owned by client u_i in communication round t are denoted by $\mathcal{W}_i^t$ with W layers. For a given mini-batch $(\boldsymbol{x}_i, y_i)$ sampled from $\mathcal{D}_i$, the output of the model is denoted by $\mathcal{W}_i^t(\boldsymbol{x}_i) = \mathcal{W}_{i,(0,W-1)}^t(\boldsymbol{x}_i) = \mathcal{W}_{i,(W-1)}^t(\mathcal{W}_{i,(W-2)}^t(\cdots \mathcal{W}_{i,(0)}^t(\boldsymbol{x}_i)))$, where $\mathcal{W}_{i,(j,k)}^t$ denotes layers j to k of model $\mathcal{W}_i^t$, $\mathcal{W}_{i,(j)}^t$ denotes layer j of model $\mathcal{W}_i^t$, $\boldsymbol{x}_i$ denotes the input samples, and y_i denotes the corresponding labels. The backward propagation can be represented in the similar way: $\tilde{\mathcal{W}}_i^t(grad) = \tilde{\mathcal{W}}_{i,(0,W-1)}^t(grad) = \tilde{\mathcal{W}}_{i,(0)}^t(\tilde{\mathcal{W}}_{i,(1)}^t(\cdots \tilde{\mathcal{W}}_{i,(W-1)}^t(grad)))$, where $\tilde{\mathcal{W}}_i^t$ denotes the backward propagation process of $\mathcal{W}_i^t$.

3.2.1 *Initialization*

In the beginning, each client uploads its state information (C_i, D_i) to the server, where C_i and D_i are the computa-

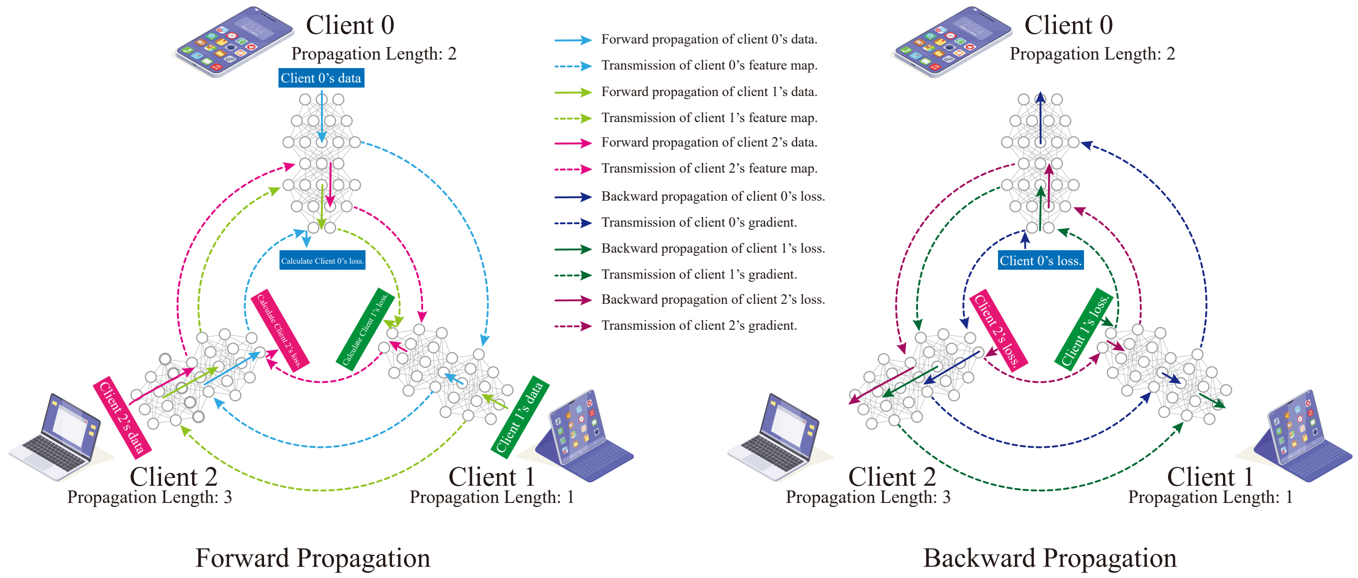

Fig. 2. Forward and backward propagation processes for RingSFL with 3 clients. A multilayer perceptron (MLP) containing 6 fully connected layers is trained, and the propagation length is set to: $L_0 : L_1 : L_2 = 2 : 1 : 3$.

tional power and dataset size of u_i, respectively. The server determines the propagation length $L_i = \frac{C_i}{\sum_{j=0}^{N-1} C_j} W$ and aggregation weight $a_i = \frac{D_i}{\sum_{j=0}^{N-1} D_j}$ for each client based on their state information. Finally, the server dispatches (L_i, a_i) to the corresponding client u_i along with the initialized global model $\mathcal{W}^0$. The derivation of the propagation length is described in detail in subsection 3.3. And the aggregation weights are set in the same way as FedAvg [5], as this is not the focus of this paper.

3.2.2 Forward Propagation

In each communication round, forward and backward propagation are conducted multiple times, and the training processes of different clients are executed in parallel. The forward propagation process of client u_i can be divided into three phases: *Starting Phase*, *Relay Phase*, and *Stop Phase*.

- `Starting Phase`: Client u_i samples a mini-batch of input data $(\boldsymbol{x}_i, y_i)$ from its local dataset $\mathcal{D}_i$, feeds $\boldsymbol{x}_i$ into local model to get the middle feature map $\bar{\boldsymbol{x}}_i = \mathcal{W}^t_{i,(0,L_i-1)}(\boldsymbol{x}_i)$. Then, u_i sends $(\bar{\boldsymbol{x}}_i, l_{stop})$ to the next client u_{i+1} in the ring topology, where $l_{stop} = L_i - 1$ denotes the index of the output layer.
- `Relay Phase`: For the next clients in the ring $\{u_{i+1}, \cdots, u_{N-1}, u_0, \cdots, u_{i-1}\}$, their job is to relay the feature maps from the previous client. Specifically, for each client $u_j \in \{u_{i+1}, \cdots, u_{N-1}, u_0, \cdots, u_{i-1}\}$, they receive $(\bar{\boldsymbol{x}}_i, l_{stop})$ from the previous client u_{j-1}, feed it into the local model $\mathcal{W}^t_{j,(l_{stop}+1, l_{stop}+L_j)}$ to obtain a new feature map $\bar{\boldsymbol{x}}_i = \mathcal{W}^t_{j,(l_{stop}+1, l_{stop}+L_j)}(\bar{\boldsymbol{x}}_i)$ and send $(\bar{\boldsymbol{x}}_i, l_{stop} + L_j)$ to the next client u_{j+1}.
- `Stop Phase`: Finally, client u_i will receive the corresponding model output $\bar{\boldsymbol{x}}_i$ from u_{i-1}, and the $loss = loss\_fn(\bar{\boldsymbol{x}}_i, y_i)$ is calculated based on it.

3.2.3 Backward Propagation

The backward propagation process is similar to the forward propagation and can be divided into three phases: *Starting Phase*, *Relay Phase*, and *Stop Phase*.

- `Starting Phase`: Client u_i takes the $loss$ calculated locally as the start point of backward propagation, sends $(grad, l_{stop}, a_i)$ to the previous client u_{i-1} along the ring topology, where $grad = loss, l_{stop} = W - 1$, and $a_i = \frac{D_i}{\sum_{j=0}^{N-1} D_j}$.
- `Relay Phase`: For the remaining clients in the ring $\{u_{i-1}, \cdots, u_0, u_{N-1}, \cdots, u_{i+1}\}$, their job is to relay the gradients from the previous client. Specifically, for each client $u_j \in \{u_{i-1}, \cdots, u_0, u_{N-1}, \cdots, u_{i+1}\}$, they receive $(grad, l_{stop}, a_i)$ from u_{j+1}, back propagate L_j layers in $\mathcal{W}^t_{j,(l_{stop}-L_j+1, l_{stop})}$ to get the corresponding gradient $\boldsymbol{g}^t_{j,(l_{stop}-L_j+1, l_{stop})}$, which will be weighted by a_i and cached. Then, u_j sends $(\boldsymbol{g}^t_{j,(l_{stop}-L_j+1)}, l_{stop} - L_j, a_i)$ to u_{j-1}, where $\boldsymbol{g}^t_{j,(l_{stop}-L_j+1)} = \mathcal{W}^t_{j,(l_{stop}-L_j+1, l_{stop})}(grad)$.
- `Stop Phase`: Finally, the backward propagation will stop at client u_i, and the gradients are weighted by its aggregation weight a_i and cached in all clients separately. Based on the gradients cached locally, each client updates their local model.

3.2.4 Model Aggregation

In each communication round, the trained local model parameters $\mathcal{W}_i^{t+1}$ are uploaded to the server for aggregation. Since the gradients are already weighted during the training process, model aggregation can be achieved by direct averaging: $\mathcal{W}^{t+1} = \frac{1}{N} \sum_{i=0}^{N-1} \mathcal{W}_i^{t+1}$, which will be detailed in section 3.4. RingSFL performs multiple communication rounds until the model converges. It can be seen from the forward/backward propagation that the local model of a client is trained by all clients, which leads to a blended

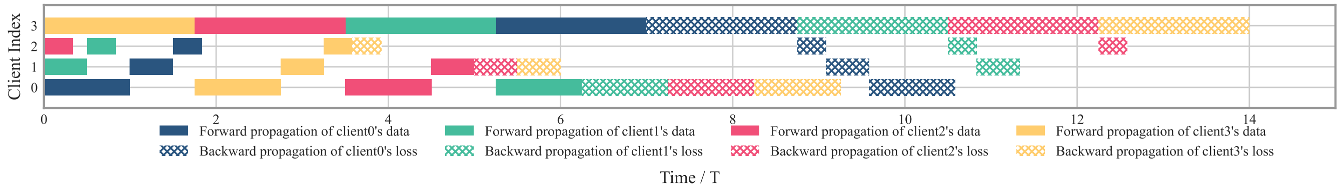

Fig. 3. Propagation flow of RingSFL with $\frac{C_0}{C}=0.1, \frac{C_1}{C}=0.2, \frac{C_2}{C}=0.3, \frac{C_3}{C}=0.4, p_0=0.1, p_1=0.1, p_2=0.1, p_3=0.7$.

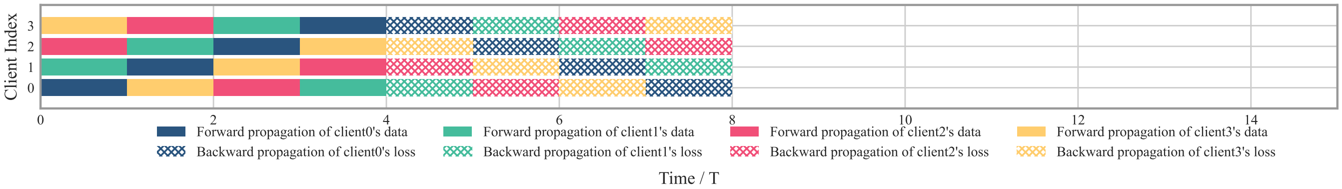

Fig. 4. Propagation flow of RingSFL with $\frac{C_0}{C}=p_0=0.1, \frac{C_1}{C}=p_1=0.2, \frac{C_2}{C}=p_2=0.3, \frac{C_3}{C}=p_3=0.4$.

model. This poses increasing difficulty in training data reconstruction from the eavesdropped parameters.

3.2.5 Client Dropout Handling

In RingSFL, due to the instability of wireless communication or client battery depletion, etc., there may exist some clients suddenly dropping out of training in the middle of the training process, resulting in the ring topology disconnection. Fortunately, since the server acts as the controller of the system, it can re-organize the ring before the following round of training. This limits the impact of client dropping to only one training round instead of the whole training process. Specifically, when a client suddenly drops from the ring, the server will immediately detect which client is dropped. Then, the server will stitch the ring back together from the breakpoint (i.e., control the two clients at the breakpoint to re-establish the communication connection) and reconfigure propagation lengths for each client that is still online in order to continue the training, while the client that drops out will be ignored.

3.3 Model Split Scheme

In order to minimize the training time in RingSFL, model splitting is implemented at the logical level by allocating different propagation lengths L_i to each client. This subsection details the procedure for determining L_i.

To minimize the training time, the training load of each client should match its computational power. We denote the computation required to complete training for a mini-batch (including forward propagation, backward propagation and parameter updates) by M GFLOPs, then the total computation required to complete a mini-batch of training for all users in the system is MN. And the computation undertaken by u_i can be denoted by p_iMN, where p_i denotes the ratio of the training load assigned to u_i to the total training load, $\sum_{i=0}^{N-1} p_i = 1$. Then, the computation time consumed by u_i to complete the training of a mini-batch is $\frac{p_iMN}{C_i}$.

Since there are N clients in the system, the computation time consumed by the straggler to complete the training of a mini-batch is $\max\left\{\frac{p_0MN}{C_0}, \frac{p_1MN}{C_1}, \cdots, \frac{p_{N-1}MN}{C_{N-1}}\right\}$. To minimize the time consumption of the straggler, we formulate the following optimization problem.

$$\min_{p_0,\cdots,p_{N-1}} \max\left\{\frac{p_0MN}{C_0}, \frac{p_1MN}{C_1}, \cdots, \frac{p_{N-1}MN}{C_{N-1}}\right\} \tag{1}$$

$$\text{s.t.} \quad \sum_{i=0}^{N-1} p_i = 1, \tag{1a}$$

$$0 \le p_i \le 1, \quad \forall i = 0, \cdots, N-1. \tag{1b}$$

Problem (1) is a MinMax problem. By introducing a new variable m, it can be rewritten as

$$\min_{p_0,\cdots,p_{N-1},m} \quad m \tag{2}$$

$$\text{s.t.} \quad \frac{p_iMN}{C_i} - m \le 0, \quad \forall i = 0, \cdots, N-1, \tag{2a}$$

$$\sum_{i=0}^{N-1} p_i = 1, \tag{2b}$$

$$0 \le p_i \le 1, \quad \forall i = 0, \cdots, N-1. \tag{2c}$$

Now, the original problem has been transformed into an ordinary linear programming problem, and it is easy to find the optimal solution as

$$\begin{cases} p_i^* = \dfrac{C_i}{\sum_{j=0}^{N-1} C_j}, & \forall i = 0, \cdots, N-1, \\ m^* = \dfrac{MN}{\sum_{j=0}^{N-1} C_j}. \end{cases} \tag{3}$$

The solution reveals that the optimal p_i^* should be equal to the ratio of the computational power $\frac{C_i}{\sum_{j=0}^{N-1} C_j}$. Since the training load of each client can be controlled by the number of locally trained neural layers, we set the propagation length to $L_i = p_i^* W = \frac{C_i}{\sum_{j=0}^{N-1} C_j} W$. However, in practice, the W layers of the model are of different sizes, and $p_i^* W$ is generally not an integer. Therefore, the server should adjust the propagation length L_i accordingly. This will affect the total training time, and we leave the details for future work.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3309633

To clearly show the relationship between the total training time and the training load of each client, Fig. 3 illustrates the computation flow of each client in a 4-client RingSFL system with $\frac{C_0}{C} = 0.1$, $\frac{C_1}{C} = 0.2$, $\frac{C_2}{C} = 0.3$, $\frac{C_3}{C} = 0.4$ and $p_0 = 0.1, p_1 = 0.1, p_2 = 0.1, p_3 = 0.7$, where $C = \sum_{i=0}^{N-1} C_i$ denotes the total computational power of all clients in the system. The unit of time axis in the figure is set to $T = \frac{M}{2C}$. The figure shows that although client 3 has more computational power than the other clients, it becomes a straggler in training due to the excessive training load. As a result, the time cost of the system to complete a mini-batch training is bottlenecked by this client, who spends the longest training time. Therefore, the computational load should be optimally allocated to suppress the straggler effect so that the training time is minimized.

Fig. 4 shows the computation flow of each client in a 4-client RingSFL system with $p_0 = \frac{C_0}{C} = 0.1, p_1 = \frac{C_1}{C} = 0.2, p_2 = \frac{C_2}{C} = 0.3, p_3 = \frac{C_3}{C} = 0.4$. From the figure, it can be seen that the straggler effect is significantly mitigated by optimally allocating p_i. The computation time consumed by each client is equal, and the time consumed by the system to complete a mini-batch training is reduced to 8 units of time.

3.4 Model Aggregation Scheme

The server receives local models from clients, which are then aggregated. Due to the blended model, a revised model aggregation scheme in RingSFL is necessary because the conventional FedAvg algorithm [5] is inapplicable.

Different from FedAvg, the weighting in RingSFL is achieved by each client during the training process. The aggregation weight a_i is transferred among clients along with backward propagation, and the computed gradients are weighted by it. To keep the formulation concise, we assume that each client's local dataset has only one mini-batch and only one epoch is trained for each communication round, while the results of multiple mini-batch updates can be easily generalized. Since each neural layer of a client may have multiple propagation flows passing through it, we use $\mathcal{U}_{i,(j)}$ to denote the set of clients that propagate through $\mathcal{W}_{i,(j)}$, where $\bigcup_{i=0}^{N-1} \mathcal{U}_{i,(j)} = \{0, \cdots, N-1\}$. Then, the local training result for u_i in the t-th communication round is

$$\mathcal{W}_i^{t+1} = \begin{bmatrix} \mathcal{W}_{i,(0)}^t - \sum_{j\in\mathcal{U}_{i,(0)}} \eta a_j \boldsymbol{g}_{j,(0)}^t \\ \vdots \\ \mathcal{W}_{i,(k)}^t - \sum_{j\in\mathcal{U}_{i,(k)}} \eta a_j \boldsymbol{g}_{j,(k)}^t \\ \vdots \\ \mathcal{W}_{i,(W-1)}^t - \sum_{j\in\mathcal{U}_{i,(W-1)}} \eta a_j \boldsymbol{g}_{j,(W-1)}^t \end{bmatrix}, \tag{4}$$

where $\boldsymbol{g}_{j,(k)}^t$ is the gradient of the k-th layer calculated based on the data from u_j's dataset in communication round t, a_j is aggregation weight of u_j, and η is learning rate. Server can aggregate by simply averaging after receiving $\mathcal{W}_i^{t+1}$, $\forall i = 0, \cdots, N-1$. The new global model obtained by server aggregation is

$$\begin{aligned} \mathcal{W}^{t+1} &= \frac{1}{N}\sum_{i=0}^{N-1} \mathcal{W}_i^{t+1} \\ &= \begin{bmatrix} \mathcal{W}_{i,(0)}^t - \frac{\eta}{N}\sum_{i=0}^{N-1}\sum_{j\in\mathcal{U}_{i,(0)}} a_j \boldsymbol{g}_{j,(0)}^t \\ \vdots \\ \mathcal{W}_{i,(k)}^t - \frac{\eta}{N}\sum_{i=0}^{N-1}\sum_{j\in\mathcal{U}_{i,(k)}} a_j \boldsymbol{g}_{j,(k)}^t \\ \vdots \\ \mathcal{W}_{i,(W-1)}^t - \frac{\eta}{N}\sum_{i=0}^{N-1}\sum_{j\in\mathcal{U}_{i,(W-1)}} a_j \boldsymbol{g}_{j,(W-1)}^t \end{bmatrix}. \end{aligned} \tag{5}$$

Since $\bigcup_{i=0}^{N-1} \mathcal{U}_{i,(k)} = \{0, \cdots, N-1\}$, we can derive that $\sum_{i=0}^{N-1}\sum_{j\in\mathcal{U}_{i,(k)}} a_j \boldsymbol{g}_{j,(k)}^t = \sum_{i=0}^{N-1} a_i \boldsymbol{g}_{i,(k)}^t$. Then, (5) can be rewritten as

$$\begin{aligned} \mathcal{W}^{t+1} &= \begin{bmatrix} \mathcal{W}_{i,(0)}^t - \frac{\eta}{N}\sum_{i=0}^{N-1} a_i \boldsymbol{g}_{i,(0)}^t \\ \vdots \\ \mathcal{W}_{i,(k)}^t - \frac{\eta}{N}\sum_{i=0}^{N-1} a_i \boldsymbol{g}_{i,(k)}^t \\ \vdots \\ \mathcal{W}_{i,(W-1)}^t - \frac{\eta}{N}\sum_{i=0}^{N-1} a_i \boldsymbol{g}_{i,(W-1)}^t \end{bmatrix} \\ &= \mathcal{W}^t - \frac{\eta}{N}\sum_{i=0}^{N-1} a_i \begin{bmatrix} \boldsymbol{g}_{i,(0)}^t \\ \vdots \\ \boldsymbol{g}_{i,(k)}^t \\ \vdots \\ \boldsymbol{g}_{i,(W-1)}^t \end{bmatrix} \\ &= \mathcal{W}^t - \frac{\eta}{N}\sum_{i=0}^{N-1} a_i \boldsymbol{g}_i^t. \end{aligned} \tag{6}$$

The formulation of the aggregation result in (6) is similar to FedAvg [5], except that the learning rate is reduced by N times. To compensate for the discounted learning rate, we can manually multiply the learning rate by the number of clients participating in the training.

3.5 Overlapping Layers Can Improve Model Performance

As discussed previously, since each client has different computation resources, it has different propagation lengths and further leads to the presence of overlapping layer. We first give the definition of overlapping layer and then discuss the properties of overlapping layer and their impact on the performance of the global model.

Overlapping Layer: If a neural layer is propagated by multiple clients' propagation flow, we call that layer an overlapping layer.

As illustrated in Fig. 5, we consider two clients collaboratively training a multilayer perceptron (MLP) with 6 fully connected layers using RingSFL. The local models of client 0 and client 1 are denoted by $\mathcal{W}_0^t$ and $\mathcal{W}_1^t$, and the propagation lengths are set to $L_0 = 2$ and $L_1 = 4$. The

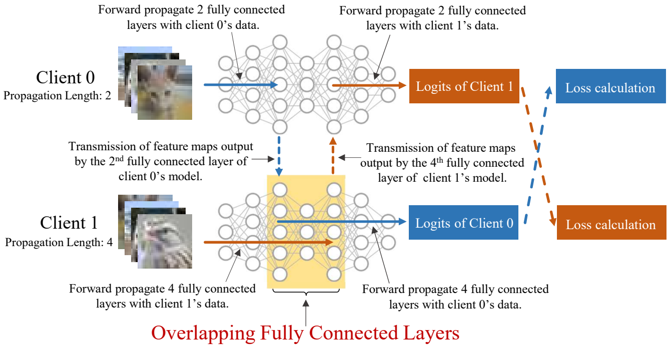

Fig. 5. Forward propagation processes for RingSFL with 2 clients. A multilayer perceptron (MLP) containing 6 fully connected layers is trained, and the propagation length is set to: $L_0 : L_1 = 2 : 4$.

propagation flow of client 0 is illustrated by the blue arrows, which propagate through $\mathcal{W}^t_{0,(0,1)}$ and $\mathcal{W}^t_{1,(2,5)}$, respectively, and the propagation flow of client 1 is illustrated by the red arrows, which propagate through $\mathcal{W}^t_{1,(0,3)}$ and $\mathcal{W}^t_{0,(4,5)}$, respectively. It can be noted that $\mathcal{W}^t_{1,(2,3)}$ is contained by both $\mathcal{W}^t_{1,(2,5)}$ and $\mathcal{W}^t_{1,(0,3)}$, i.e., the propagation flows of both client 0 and client 1 pass over $\mathcal{W}^t_{1,(2,3)}$ (with both blue and red arrows passing over them in Fig. 5), so we call $\mathcal{W}^t_{1,(2,3)}$ the overlapping fully connected layers in this RingSFL system. During backpropagation, the gradients of both client 0 and client 1 for the 3-rd and 4-th fully connected layers will accumulate to the overlapping fully connected layers, which means that the gradients of both client 0 and client 1 for the 3-rd and 4-th fully connected layers will be aggregated at each backpropagation. Since the overlapping fully connected layers have a higher aggregation frequency compared to other layers, their gradients are more capable of driving the parameters of the overlapping fully connected layers toward the global optimum, which can effectively improve the performance of the global model.

Considering that the overlapping layer has a higher aggregation frequency and its gradient is better than that of other layers, we try to increase the step size of the overlapping layer when updating parameters (or increase the learning rate of the overlapping layer) to better utilize the properties of the overlapping layer, and thus propose **RingSFLv2**[1]. Specifically, we multiply the learning rate of the overlapping layer with the number of overlaps of the layer so that the learning rate is proportional to the number of overlaps to adjust the update step size of the overlapping layer. If the j-th layer of the local model of the client u_i is an overlapping layer, the parameter update process for this overlapping layer $\mathcal{W}_{i,(j)}$ can be formulated as

$$\mathcal{W}^t_{i,(j)} = \mathcal{W}^t_{i,(j)} - \eta|\mathcal{U}_{i,(j)}| \sum_{k \in \mathcal{U}_{i,(j)}} a_k \boldsymbol{g}^t_{k,(j)}, \tag{7}$$

where $\mathcal{U}_{i,(j)}$ denotes the set of clients whose propagation flow passes through $\mathcal{W}_{i,(j)}$, the number of clients in this set is denoted by $|\mathcal{U}_{i,(j)}|$, and η denotes the default learning rate. For example, in Fig. 5, the overlapping fully connected layers $\mathcal{W}^t_{1,(2,3)}$ has two overlaps (with propagation flows of two clients passing through), and then its learning rate is 2η. By improving the update method of the overlapping layer, we find that the performance of the global model is effectively improved, which is discussed in detail in the experimental section.

3.6 Privacy Enhancement

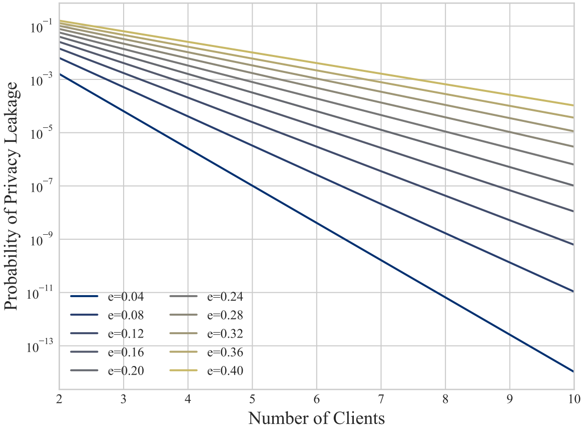

Fig. 6. Impact of the number of clients and the probability of communication links being eavesdropped on the probability of privacy leakage.

While a theoretical proof of privacy is beyond the scope of this paper, this subsection will provide a brief overview of the privacy-enhancing features of RingSFL. There are two potential routes to accomplish the attack on RingSFL: **Reassembling blended models**: reassembling the complete model belonging to each client by eavesdropping on all blended models sent to the server and obtaining the sensitive information carried in the reassembled model. **Model inversion attack**: recovering the training data of each client by eavesdropping on the D2D communication links between the clients. We discuss these two cases respectively.

3.6.1 *Reassembling Blended Models*

According to existing research [10], [11], an attacker must obtain the complete model parameters or gradients to recover client data, and cannot recover from only partial or broken models. Since clients upload blended models to the server, an attacker (an eavesdropper or a malicious aggregation server) must reassemble these blended models based on propagation lengths to obtain the complete models belonging to each client. However, in RingSFL system, there are usually overlapping layers where multiple clients' model parameters are accumulated, and recovering individual clients' model parameters from these overlapping layers is quite difficult, so RingSFL still provides enhanced privacy.

A special case worth noting is that no overlapping layer exists in the RingSFL system when the propagation lengths of all clients are equal, and the possibility of privacy leakage exists at this point. We further discuss the probability of privacy leakage in this case. Using e_i to denote the probability that the communication link between u_i and the server is eavesdropped, the probability of privacy leakage can be expressed as

$$P = \prod_{i=0,\cdots,N-1} e_i. \tag{8}$$

---

[1] RingSFL without learning rate adjustment is referred to as RingSFLv1.

It can be seen from (8) that the privacy leakage probability is influenced by the eavesdropping probability e_i and the number of clients N. Fig. 6 illustrates the effect of different eavesdropping probabilities and the number of clients on the privacy leakage probability, where $e = e_i, \quad \forall i \in 0, \cdots, N-1$. It can be seen that the probability of privacy leakage decreases exponentially with the increase in the number of clients. Even with a high eavesdropping probability on each link (e.g., $e = 0.4$), the leakage probability still decreases rapidly to a value close to zero. This means that when the number of clients in the RingSFL system is large enough, a high level of security can be maintained even in particularly extreme cases (equal propagation length per client with a high eavesdropping probability).

3.6.2 Model Inversion Attack

Since RingSFL introduces D2D communication links between clients to transmit feature maps, an eavesdropper may try to recover the original sensitive data of the client from the eavesdropped feature maps through model inversion attack techniques. We therefore discuss the privacy performance of RingSFL under different model inversion attack settings. As discussed in [39], there are three main settings for the model inversion attack: **white-box setting**, **black-box setting**, and **query-free setting**, and we discuss each of these three settings.

White-box Setting: In this setting, there are two main prerequisites for a successful attack: 1. the attacker needs to know the model structure and parameters of the attacked client; 2. the attacker needs to obtain the feature maps of the local dataset of the attacked client. Once these two prerequisites are satisfied, the attacker will first randomly generate dummy data samples and feed them into a model with the same parameters as the attacked client to obtain the feature maps of these dummy data samples. Subsequently, the attacker optimizes the generated dummy data samples by minimizing the distance between the feature maps of the dummy data samples and the feature maps of the real samples (obtained from the attacked client), and finally recovers the real data samples.

In the white-box setting, the attacker needs to obtain the local model parameters and the output feature maps of the attacked client. For attackers without eavesdropping capabilities (e.g., malicious participating clients or malicious servers), there is no privacy leakage problem in this setting because malicious participating clients do not have access to the local model parameters of other clients, and malicious servers do not have access to the client's output feature maps. However, RingSFL still has limitations for attackers with strong eavesdropping capabilities and may not be effective against white-box attacks. An attacker can perform a white-box attack by eavesdropping on the communication link between the attacked client and the server to obtain the local model parameters of the attacked client, and by eavesdropping on the D2D communication link between the attacked client and its neighboring clients to obtain the feature maps of the samples in the training dataset of the attacked client. One potential defense scheme [40] is to reduce the mutual information between the feature maps and the real samples by adding regular term to the local objective function at training time, thus making it more difficult for the attacker to recover the real samples. This approach solely involves modifying the client's local objective function (or local loss function), thereby ensuring no disruption to the RingSFL training process. However, it is important to note that decreasing the mutual information may adversely affect the accuracy of the global model. Thus, achieving an optimal balance between global model accuracy and privacy performance necessitates careful consideration and trade-offs.

Black-box Setting: In this setting, a successful attack requires that the attacker is able to input locally generated dummy data samples into the local model of the attacked client for inference and obtain the corresponding feature maps. Once this prerequisite is satisfied, the attacker randomly generates dummy data samples and inputs them into the local model of the attacked client for inference, and obtains the feature maps of the dummy data samples. Subsequently, the attacker uses these dummy data samples and their corresponding feature maps as a dataset to train a model that can replicate the behavior of the attacked client's local model. Eventually, by using this trained model, the model inversion attack can be executed using the same approach as in the white-box setting.

However, during the training process of RingSFL, the clients only access local datasets and do not receive any external datasets nor provide query services, and the attacker cannot input the generated dummy data samples into the local model of the attacked client. Therefore, the prerequisites for a successful attack cannot be met, making RingSFL effective against model inversion attacks under black-box settings.

Query-free Setting: In this setting, a successful attack requires that the attacker must have knowledge of the local dataset distribution of the attacked client. This setting assumes that the attacker cannot directly query the local model of the attacked client or access its internal parameters. Instead, the attacker relies on generating dummy data samples with the same distribution as the local dataset of the attacked client. The attacker then uses these generated samples to train a model that mimics the behavior of the attacked client's local model. Finally, based on this trained model, the attacker performs a model inversion attack using the same approach as in the white-box setting.

However, during the training process of RingSFL, the clients neither transmit their local datasets nor the distribution information of their local datasets. Therefore, the necessary prerequisites for a successful attack cannot be met, allowing RingSFL to effectively resist model inversion attacks under query-free setting.

4 EXPERIMENTAL RESULTS

In this section, we assess the efficacy of the proposed scheme and analyze the performance improvements. We commence by introducing the experimental configuration and then present the experimental results.

4.1 Experimental Setup

4.1.1 Platforms

Our proposed RingSFL scheme was evaluated in both a simulated environment and a real prototype system.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3309633

Algorithm 1: RingSFL Training Process

```
Server initializes model parameters $\mathcal{W}^0$ with $W$ layers;
Clients upload state information ($D_i$: dataset size, $C_i$: computational power) to the server;
Server calculates propagation length and aggregation weight for clients:
    $L_i = p_i W = \frac{C_i}{\sum_{j=0}^{N-1} C_j} W;\ a_i = \frac{D_i}{\sum_{j=1}^{N} D_j}$;
Server sends $(\mathcal{W}^0, L_i, a_i)$ to each client $u_i$;
for each round $t$ from 0 to $T-1$ do
    for each client $u_i$ parallelly do
        Set local model $\mathcal{W}_i^t = \mathcal{W}^t$;
        for each epoch from 0 to $E-1$ do
            for each batch $(\boldsymbol{x}_i, y_i)$ in $\mathcal{D}_i$ do
                BatchUpdate($\mathcal{W}_i^t$, $(\boldsymbol{x}_i, y_i)$);
            end
        end
        Upload training results $\mathcal{W}_i^{t+1}$;
    end
    Server aggregates parameters:
        $\mathcal{W}^{t+1} = \frac{1}{N}\sum_{i=0}^{N-1} \mathcal{W}_i^{t+1}$;
    Server sends $\mathcal{W}^{t+1}$ to each client;
end
```

Algorithm 2: BatchUpdate($\mathcal{W}_i^t$, $(\boldsymbol{x}_i, y_i)$)

```
$\bar{\boldsymbol{x}}_i = \boldsymbol{x}_i, l_{stop} = -1$;
for $u_j$ in $[u_i, \cdots, u_{N-1}, u_0, \cdots, u_{i-1}]$ do
    $\bar{\boldsymbol{x}}_i = \mathcal{W}^t_{j,(l_{stop}+1, l_{stop}+L_j)}(\bar{\boldsymbol{x}}_i)$;
    $l_{stop} = l_{stop} + L_j$;
    Send $(\bar{\boldsymbol{x}}_i, l_{stop})$ to $u_{j+1}$;
end
$loss = loss\_fn(\bar{\boldsymbol{x}}_i, y_i)$;
$grad = loss,\ l_{stop} = W - 1$;
for $u_j$ in $[u_{i-1}, \cdots, u_0, u_{N-1}, \cdots, u_i]$ do
    $grad = \tilde{\mathcal{W}}^t_{j,(l_{stop}-L_j+1, l_{stop})}(grad)$;
    Cache gradients $a_i \cdot \boldsymbol{g}^t_{j,(l_{stop}-L_j+1, l_{stop})}$ calculated in previous step;
    $l_{stop} = l_{stop} - L_j$;
    Send $(grad, l_{stop}, a_i)$ to $u_{j-1}$;
end
Each client $u_i$ updates their local model $\mathcal{W}_i^t$ based on the gradients cached;
```

Simulation Environment: The software used in the simulation was written in Python 3.9.12, and PyTorch 1.11.0 was employed for model construction and training to ensure compatibility with the software environment of the prototype system. In the simulation, a ring topology was formed by 5 clients, who then conducted RingSFL with a server. The model performance of RingSFL was evaluated using different models and both IID and Non-IID datasets.

Prototype System: As depicted in Fig. 7, our prototype system consists of three Raspberry Pi nodes (ARM Cortex-A72 @ 1.5GHz 6.4W) as weak computational power clients and two PC nodes (11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GHz 65W) as strong computational power clients. A laptop is utilized as a parameter server for model aggregation. Each device operates at the 5440 MHz band with a channel of 40 MHz bandwidth, allowing D2D TCP rates of 135 Mbps with a standard deviation of 5.83. Through the prototype system, we mainly assess the capability of RingSFL to adapt to the client heterogeneity in real-world settings in terms of training time and energy consumption. As it is difficult to measure the energy consumption in the prototype system, we instead capture each client's actual CPU occupation time and convert it into consumed energy by multiplying it by the single-core full power. Additionally, we evaluate the effect of D2D communications on the proposed RingSFL scheme.

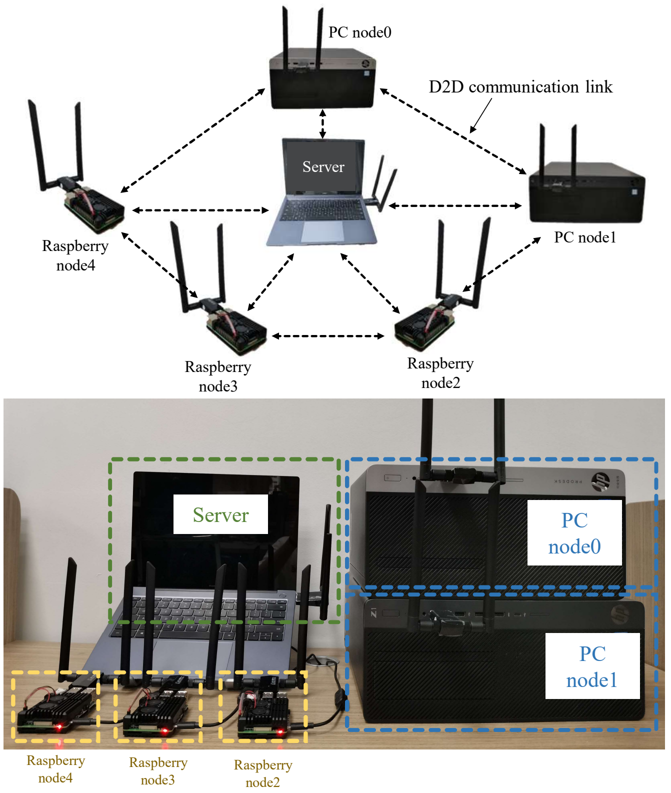

Fig. 7. Prototype system of RingSFL with 2 PC nodes and 3 raspberry Pi nodes.

4.1.2 Datasets and Models

Datasets: The experiments utilize the widely-used MNIST [41] and CIFAR10 [42] datasets. Specifically, MNIST consists of 70,000 square (28x28=784 pixel) grayscale handwritten digital images, divided into 10 categories (60,000 for training and 10,000 for testing); Cifar10 consists of 60,000 square (3x32x32=3072 pixel) 3-channel color images, divided into 10 categories (50,000 for training and 10,000 for testing). Each client's local dataset has the same number of samples. Furthermore, to evaluate the effect of the dataset distribution on RingSFL, experiments are conducted using both IID and non-IID data. Specifically, the Non-IID dataset is constructed by assigning each client a sub-dataset containing only two categories.

Models: To validate the performance of our scheme, we employ mainstream networks for evaluation, including ResNet18 [43], VGG16 [44], AlexNet [45], and LeNet-5 [41].

To ensure the model converges to the desired accuracy, we initialize the model parameters prior to training. Specifically, the Conv2d layer is initialized using the Kaiming normal method. The weights of the BatchNorm2d layer are initialized to 1, and the bias is initialized to 0. The weights of the Linear layer are initialized using a Gaussian distribution with a mean of 0 and standard deviation of 0.01, and the bias is initialized to 0.

4.1.3 Implementation

TABLE 2
Implementation Parameters.

Model	Dataset	Learning Rate (IID / Non-IID)	Block Num
ResNet18	CIFAR10	0.1 / 0.04	10
VGG16	CIFAR10	0.01 / 0.0015	16
AlexNet	MNIST	0.003 / 0.003	17
LeNet-5	MNIST	0.02 / 0.02	12

In our experiments, we use vanilla FL [5], vanilla SL [26], and SplitFed [30] as benchmarks. For each experiment, five clients (2 PC nodes u_0, u_1 and 3 raspberry Pi nodes u_2, u_3, u_4) are included for 100 training rounds. In each round, each client iterates two epochs on their local dataset. Since vanilla SL has no parameter aggregation process, the whole training process cannot be directly divided into rounds. Therefore, in our experiments, when vanilla SL updates $local\_epoch\_num \times batch\_num$ steps, we treat it as a round for a fair comparison with other algorithms[2]. SGD optimizer is used for stochastic gradient descent to optimize the model. For the convenience of the logical level model splitting, we pre-divide models into blocks. Specifically, ResNet18 is divided into 10 blocks, VGG16 is divided into 16 blocks, AlexNet is divided into 17 blocks, and LeNet-5 is divided into 12 blocks. The complete training parameters are presented in Table 2.

4.2 Results and Discussion

We first evaluate the convergence performance of RingSFL and the impact of overlapping layers on model accuracy in the simulation environment. Subsequently, the impact of the D2D communication rate on RingSFL and the efficiency of RingSFL in terms of training time and energy consumption are evaluated in the prototype system. Finally, we validate the privacy enhancement achieved by RingSFL. Due to space limitations, we mainly discuss the performance of RingSFL with ResNet18 as an example. Please refer to the tables for the experimental results of other models.

4.2.1 Convergence Performance

This experiment serves to validate the convergence performance of RingSFL, with SplitFed, vanilla FL, and vanilla SL used as benchmarks for comparison. The propagation lengths of RingSFL were set to L_0:L_1:L_2:L_3:L_4=6:1:1:1:1.

As illustrated in Fig. 8, RingSFLv2 achieved the highest model accuracy on the CIFAR10 (IID) dataset after the same communication rounds, compared to the considered benchmarks. RingSFLv1 performed slightly worse than RingSFLv2, yet still converged to a higher model accuracy than vanilla FL and SplitFed. This is attributed to the fact that in RingSFL, gradients accumulated on overlapping layers are aggregated in each batch of training, thus leading to a higher capability to optimize the parameters of overlapping layers towards the global optimum, and consequently, better model performance. Furthermore, Fig. 9 demonstrates the effect of different data distributions on the convergence performance of RingSFL when training on the CIFAR10 (Non-IID) dataset. It can be observed that RingSFLv2 still maintained the best model accuracy on the Non-IID dataset compared to the considered comparison algorithms. RingSFLv1 performed similarly to vanilla FL, while vanilla SL and SplitFed had difficulty in converging on the Non-IID dataset. This indicates that RingSFL is able to adapt well to the non-IID data distribution.

The results demonstrate that RingSFL has the most optimal convergence performance in comparison to the benchmarks evaluated. Notably, the distinction in performance between RingSFLv2 and RingSFLv1 further accentuates the significance of the overlapping layers in RingSFL, which will be discussed in greater detail in the subsequent subsection. The training results of the other models are presented in Table 3.

4.2.2 Effect of Overlapping Layers

In order to evaluate the impact of overlapping layers on model accuracy, we trained models with different propagation length configurations in RingSFLv2 and gradually made the propagation length distribution uneven to increase the number of overlapping layers. Specifically, we set the propagation length to L_0:L_1:L_2:L_3:L_4={2:2:2:2:2, 3:2:2:2:1, 3:3:2:1:1, 4:3:1:1:1, 5:2:1:1:1, 6:1:1:1:1} in each experiment to observe the effect of increasing the number of overlapping layers on model accuracy.

As demonstrated in Fig. 10, the effect of overlapping layers on the model accuracy is evident. The more unevenly the propagation lengths are allocated, the more overlapping layers there are, and the higher the model accuracy is. This phenomenon can be interpreted as *the gradient reliability of overlapping layer is higher*. As discussed in [46], due to the distributed characteristic, the optimal solution of the local objective function of each client is not consistent with the optimal solution of the global objective function. Thus, the gradients computed by each client in the process of local training will not optimize the local model towards the global optimum. In the RingSFL training process, the overlapping layers accumulate the gradients from different clients, which makes the gradients on the overlapping layer more reliable and has more tendency to optimize towards the global optimum. The training results of other models are presented in Table 4. This conclusion was further validated on the Non-IID dataset, as illustrated in Fig. 11. It can be seen that this conclusion still holds for training on the Non-IID dataset.

4.2.3 Effect of Client Dropout

We further evaluated the impact of client dropout on model accuracy. In this experiment, we randomly dropped two

---

[2] $local\_epoch\_num$: The number of times the client traverses the local dataset in each communication round. $batch\_num$: The number of batches the local dataset can be divided into.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3309633

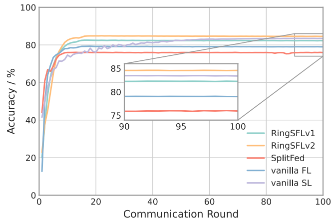

Fig. 8. Testing convergence of ResNet18 on CIFAR10 (IID) under different algorithms.

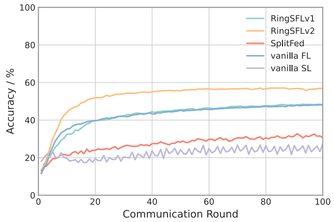

Fig. 9. Testing convergence of ResNet18 on CIFAR10 (Non-IID) under different algorithms.

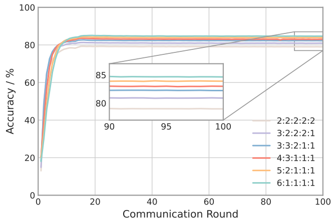

Fig. 10. Testing convergence of ResNet18 on CIFAR10 (IID) under different propagation lengths.

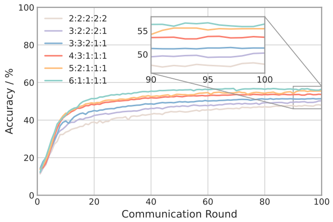

Fig. 11. Testing convergence of ResNet18 on CIFAR10 (Non-IID) under different propagation lengths.

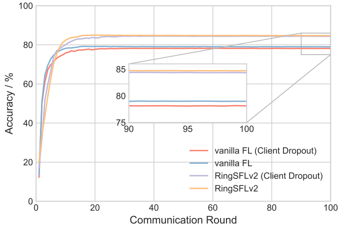

Fig. 12. Testing convergence of ResNet18 on CIFAR10 (IID) with randomly two clients dropping out in each communication round.

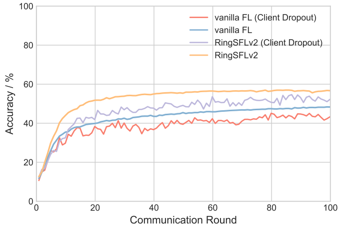

Fig. 13. Testing convergence of ResNet18 on CIFAR10 (Non-IID) with randomly two clients dropping out in each communication round.

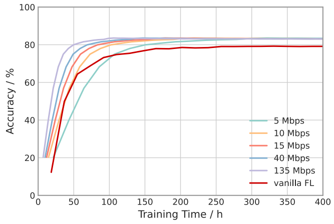

Fig. 14. Testing convergence of ResNet18 on CIFAR10 (IID) under different D2D communication rates.

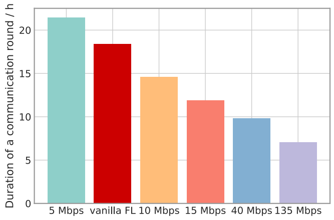

Fig. 15. Time cost of ResNet18 in a communication round under different D2D communication rates.

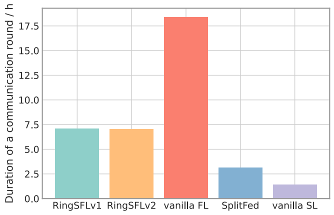

Fig. 16. Time cost of ResNet18 in a communication round under different algorithms.

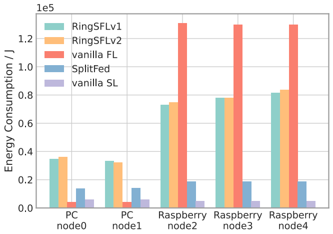

Fig. 17. Energy consumption of different devices in a communication round.

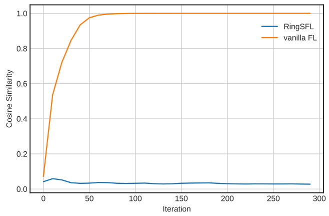

Fig. 18. Cosine similarity between reconstructed data and original data.

clients during each communication round and subsequently restored the ring topology to continue training under the server's scheduling. The CIFAR10 dataset was used, with the ResNet18 model trained, and the initial propagation lengths set to 6:1:1:1:1.

Fig. 12 showcases the training results when the local datasets are IID. It can be observed that the accuracy of the global model experiences a slight degradation when client dropout occurs during training. This is because the system loses access to the dataset of the dropped clients, resulting in a reduced amount of data available for training and subsequently impacting the overall accuracy of the global model. However, in comparison to vanilla FL, RingSFL still maintains its advantage in model accuracy. Furthermore, the effect of client dropout on RingSFL is smaller than that on vanilla FL. In addition, Fig. 13 presents the training results when the local datasets are Non-IID. Remarkably, RingSFL continues to exhibit its advantage and achieves a higher accuracy for the global model, even in the presence of Non-IID datasets.

4.2.4 Effect of D2D Communication

In order to investigate the impact of inter-user D2D communications on the RingSFL system, we varied the rates of the D2D communication links to 135 Mbps, 40 Mbps, 15 Mbps, 10 Mbps, and 5 Mbps, respectively, while keeping the communication rates between the client and the server (both RingSFL and vanilla FL) fixed at 135 Mbps. Subsequently, we evaluated the convergence of the system under these different conditions. Additionally, we conducted experiments on the CIFAR10 dataset with propagation lengths of L_0:L_1:L_2:L_3:L_4=4:3:1:1:1 using the ResNet18 model structure to further explore the impact of inter-user D2D communications. However, it is important to note that the results of this subsection are not absolute, as different model structures, datasets, and propagation lengths may lead to different outcomes.

The convergence curves of RingSFL at different D2D communication rates are depicted in Fig. 14. As the D2D communication rate decreases, the time required for RingSFL to reach the same accuracy increases accordingly. Nevertheless, it can be observed that as long as the D2D communication rate is not too bad, RingSFL can reach convergence in a shorter time compared with vanilla FL. Specifically, RingSFL takes less time to converge than vanilla FL when the D2D communication rate is greater than or equal to 10 Mbps. Conversely, when the D2D communication rate is less than or equal to 5 Mbps, RingSFL takes more time to converge than vanilla FL. This is further illustrated in Fig. 15, which shows the time consumption of RingSFL to

TABLE 3
Top-1 Accuracy (%) of Each Model under Different Algorithms. The best accuracy is marked in bold, and the secondary is marked in underline.

	ResNet18 (IID / Non-IID)	VGG16 (IID / Non-IID)	AlexNet (IID / Non-IID)	LeNet-5 (IID / Non-IID)
RingSFLv1	82.35 ± 0.36 / $\underline{48.30 \pm 0.57}$	$\underline{79.30 \pm 0.20}$ / $\underline{40.35 \pm 0.99}$	$\underline{98.83 \pm 0.11}$ / 89.58 ± 0.55	98.82 ± 0.19 / 94.34 ± 0.56
RingSFLv2	$\mathbf{84.57 \pm 0.17}$ / $\mathbf{56.80 \pm 0.78}$	$\mathbf{84.33 \pm 0.10}$ / $\mathbf{41.26 \pm 1.29}$	$\mathbf{99.13 \pm 0.07}$ / $\underline{94.31 \pm 0.88}$	$\mathbf{99.10 \pm 0.04}$ / $\underline{95.75 \pm 0.73}$
SplitFed	75.92 ± 0.51 / 30.16 ± 4.49	72.86 ± 0.62 / 28.17 ± 2.15	98.76 ± 0.09 / 84.00 ± 4.39	98.74 ± 0.24 / 93.64 ± 0.70
vanilla FL	78.93 ± 0.27 / 48.02 ± 1.28	77.02 ± 0.34 / 39.52 ± 0.81	98.81 ± 0.07 / 91.60 ± 1.14	$\underline{98.84 \pm 0.08}$ / 94.77 ± 0.29
vanilla SL	$\underline{83.41 \pm 0.44}$ / 26.96 ± 3.58	78.50 ± 0.69 / 35.33 ± 1.29	98.69 ± 0.10 / $\mathbf{98.84 \pm 0.08}$	$\underline{98.80 \pm 0.14}$ / $\mathbf{98.86 \pm 0.09}$

TABLE 4
Top-1 Accuracy (%) of Each Model under Different Propagation Lengths. The best accuracy is marked in bold, and the secondary is marked in underline.

Propagation Lengths	ResNet18 (IID / Non-IID)	Propagation Lengths	VGG16 (IID / Non-IID)	Propagation Lengths	AlexNet (IID / Non-IID)	Propagation Lengths	LeNet-5 (IID / Non-IID)
6:1:1:1:1	$\mathbf{84.66 \pm 0.33}$ / $\mathbf{56.45 \pm 1.10}$	**12:1:1:1:1**	$\mathbf{84.29 \pm 0.14}$ / 41.48 ± 1.08	**13:1:1:1:1**	99.00 ± 0.16 / $\mathbf{94.49 \pm 0.67}$	**8:1:1:1:1**	$\mathbf{99.10 \pm 0.07}$ / $\mathbf{95.85 \pm 0.32}$
5:2:1:1:1	$\underline{83.90 \pm 0.29}$ / $\underline{55.45 \pm 0.47}$	**11:2:1:1:1**	$\underline{83.98 \pm 0.24}$ / $\mathbf{42.56 \pm 0.69}$	**11:3:1:1:1**	$\underline{99.05 \pm 0.12}$ / $\underline{94.28 \pm 0.59}$	**7:2:1:1:1**	$\underline{99.04 \pm 0.06}$ / $\underline{95.79 \pm 0.30}$
4:3:1:1:1	83.00 ± 0.16 / 53.63 ± 0.62	**10:3:1:1:1**	83.78 ± 0.53 / $\underline{41.68 \pm 0.69}$	**9:5:1:1:1**	$\mathbf{99.11 \pm 0.10}$ / 93.79 ± 0.30	**6:3:1:1:1**	99.02 ± 0.05 / 95.66 ± 0.19
3:3:2:1:1	82.24 ± 0.20 / 51.34 ± 0.74	**8:3:3:1:1**	82.81 ± 0.25 / 39.25 ± 0.62	**7:5:3:1:1**	99.00 ± 0.14 / 93.00 ± 0.11	**5:3:2:1:1**	99.00 ± 0.06 / 95.65 ± 0.18
3:2:2:2:1	80.90 ± 0.19 / 50.27 ± 0.53	**6:3:3:3:1**	80.53 ± 0.32 / 37.57 ± 0.94	**5:5:3:3:1**	98.91 ± 0.07 / 92.14 ± 0.61	**4:3:2:2:1**	98.96 ± 0.04 / 95.51 ± 0.16
2:2:2:2:2	79.00 ± 0.50 / 47.89 ± 0.64	**4:3:3:3:3**	77.58 ± 0.25 / 39.76 ± 0.96	**4:4:3:3:3**	98.84 ± 0.12 / 92.53 ± 1.03	**3:3:2:2:2**	98.97 ± 0.03 / 95.45 ± 0.16

complete a training round at different D2D communication rates.

4.2.5 Convergence Time Reduction and Energy Efficiency

To further validate RingSFL's capability to accommodate system heterogeneity, we evaluate its convergence time reduction and energy efficiency on the prototype system. We utilize ResNet18 for training and set the propagation lengths of RingSFL to L_0:L_1:L_2:L_3:L_4=4:3:1:1:1.

As illustrated in Fig. 16, compared to vanilla FL, RingSFL reduces the training time by 61.6%, significantly enhancing the system's training efficiency and computational resource utilization. This is attributed to the adaptive model split scheme, which assigns more propagation lengths to clients with strong computational power and correspondingly more computational load, and vice versa. This heterogeneity-adaptive computational load balancing scheme effectively alleviates the straggler effect, thus reducing the training time. In the figure, the training time of SplitFed and vanilla SL is lower than RingSFL due to the model split mechanism, which offloads a large portion of the training tasks to the server with high computation power. Specifically, in our experiments, the client computational volume of SplitFed in each round is 1/5 of RingSFL, and the client computational volume of vanilla SL is 1/25 of RingSFL. However, they cannot achieve comparable performance in terms of model accuracy and user privacy, as discussed above.

Fig. 17 illustrates the capability of RingSFL to regulate the power consumption of the FL system. It is evident that, in comparison to vanilla FL, RingSFL increases the energy consumption of PC nodes and decreases the energy consumption of Raspberry Pi nodes, which is attributed to the ability of RingSFL to balance the computational loads among clients. This observation demonstrates that RingSFL can effectively conserve the energy of the battery-powered devices, thus enhancing the life time of the RingSFL system.

4.2.6 Privacy Preservation

To assess RingSFL's capacity to withstand data reconstruction attacks, we conducted attack experiments on both vanilla FL and RingSFL using the methods outlined in [10]. We attempted to recover the training data by eavesdropping on the model parameters transmitted between the client and the server. To measure the efficacy of the data recovery, we calculated the cosine similarity of the recovered data to the original data, which is formulated as

$$s = \frac{\boldsymbol{x}^T \hat{\boldsymbol{x}}}{\|\boldsymbol{x}\|_2 \cdot \|\hat{\boldsymbol{x}}\|_2} \tag{9}$$

where $\boldsymbol{x}$ denote the original data, $\hat{\boldsymbol{x}}$ denote the recovered data, and $\|\cdot\|_2$ denote the L2 norm of the vector. The cosine similarity between $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ can be used to measure the similarity between the two, with a value of 1 indicating perfect similarity. In this experiment, the neural network employed was LeNet and the dataset used was CIFAR10. The system was composed of five clients, and the propagation length was set to L_0:L_1:L_2:L_3:L_4=3:3:2:2:2.

The results of the data reconstruction attack are depicted in Fig. 19, where the first row corresponds to the attack on vanilla FL and the second row to the attack on RingSFL. It is evident that, after 80 iterations, the training data in vanilla FL has been successfully recovered, while the attack on RingSFL fails to recover the training data. Additionally, Fig. 18 shows that the cosine similarity between the reconstructed data and the original data has converged to 1 in the attack on vanilla FL after 80 iterations, while the cosine similarity remains around 0 in the attack on RingSFL. This is attributed to the transfer of the complete model parameters by vanilla FL, which allows the attacker to extract the complete information about the training data, thus enabling the recovery of the original training data and resulting in user privacy leakage. In contrast, the model transmitted by RingSFL is a blended model, making it difficult for an attacker to extract information about the training data from such fragmented and mixed model parameters, thus protecting user privacy effectively.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3309633

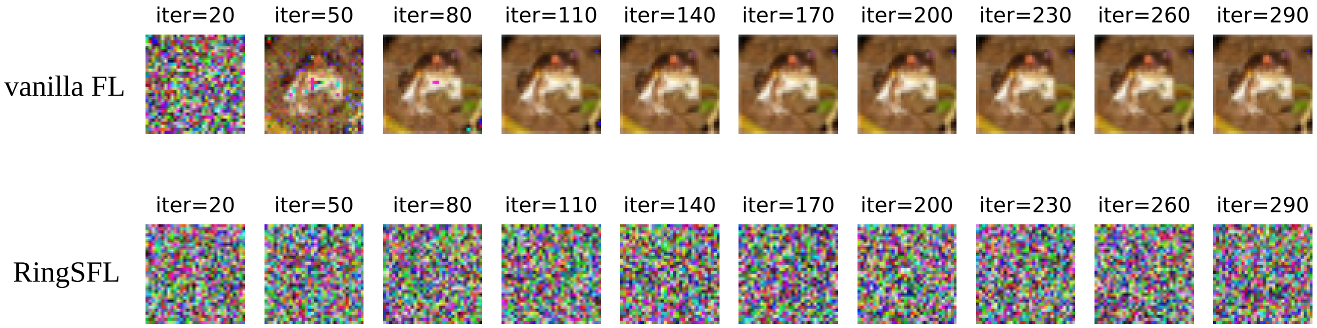

Fig. 19. Reconstructed data after attacking vanilla FL and RingSFL.

5 LIMITATIONS AND FUTURE WORKS

5.1 Incentive Mechanism Design

Fairness is a fundamental principle of multi-client collaborative training, and clients are unlikely to participate in training when they perceive it to be unfair [47]. In the RingSFL system, however, a higher computational load is assigned to clients with higher computational power, resulting in increased consumption of computation resources and battery energy. To motivate clients to participate in the training and contribute more resources, effective incentive mechanisms are necessary, such as monetary rewards. Nevertheless, verifying the actual contribution from the training results is a challenge, as each client only trains a part of the model.

5.2 Communication Cost Reduction

Compared to vanilla FL, RingSFL introduces additional D2D communication between clients. And the D2D multi-hop transmission of feature maps may make it unsuitable for scenarios with limited communication resources. One potential solution is to reduce the total amount of communication by reducing the number of clients in the ring to control the number of hops for D2D communication during forward and backward propagation. Specifically, we can split a large ring consisting of all clients into multiple subrings consisting of some clients, and perform forward and backward propagation independently in each subring. Since we can control the size of each subring, the number of hops communicated in each subring during forward and backward propagation will be reduced, and the communication cost will be significantly reduced.

5.3 Communication Topology Construction

In RingSFL, the ring topology is a critical factor in the training process and model performance. Consequently, it is essential to devise an effective algorithm for constructing the ring topology. As many factors, such as geographical location of users, channel conditions between users, the computational power of users, quality of user data sets, etc., can influence the communication topology, traditional optimization methods may not be applicable. Therefore, machine learning-based methods, such as reinforcement learning, can be explored as a potential research direction for constructing communication topologies.

5.4 Privacy Protection

When faced with a strong eavesdropper, D2D communication between clients still has the potential to compromise privacy. In addition, since the network topology is known in advance, the propagation length may reveal partial information about the system's computational resources. Therefore further design of privacy-preserving mechanisms for RingSFL is needed.

6 CONCLUSION

In this paper, we have proposed a novel FL scheme, dubbed RingSFL, which integrates FL with an efficient model split mechanism to adapt to system heterogeneity while preserving data privacy. Experiments conducted on both the simulation environment and prototype system have demonstrated that RingSFL can achieve faster convergence and higher accuracy than the benchmarks, and have better privacy performance. Furthermore, RingSFL can be applied to scenarios with significant system heterogeneity to enhance the overall system efficiency. For future works, we will design an effective incentive mechanism and learning-based optimal topology construction scheme for RingSFL.

REFERENCES

[1] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[2] D. W. Otter, J. R. Medina, and J. K. Kalita, "A Survey of the Usages of Deep Learning for Natural Language Processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604–624, 2021.

[3] J. Li *et al.*, "Recent advances in end-to-end automatic speech recognition," *APSIPA Transactions on Signal and Information Processing*, vol. 11, no. 1, 2022.

[4] Y. Wang, M. Chen, T. Luo, W. Saad, D. Niyato, H. V. Poor, and S. Cui, "Performance Optimization for Semantic Communications: An Attention-Based Reinforcement Learning Approach," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 9, pp. 2598–2613, 2022.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. International Conference on Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[6] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated Learning in Mobile Edge Networks: A Comprehensive Survey," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3309633

[7] Z. Yang, M. Chen, K.-K. Wong, H. V. Poor, and S. Cui, "Federated learning for 6G: Applications, challenges, and opportunities," *Engineering*, vol. 8, pp. 33–41, 2022.

[8] C. Yang, M. Xu, Q. Wang, Z. Chen, K. Huang, Y. Ma, K. Bian, G. Huang, Y. Liu, X. Jin *et al.*, "FLASH: Heterogeneity-Aware Federated Learning at Scale," *IEEE Transactions on Mobile Computing*, 2022.

[9] N. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/Aerial-Assisted Computing Offloading for IoT Applications: A Learning-Based Approach," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1117–1129, 2019.

[10] L. Zhu, Z. Liu, and S. Han, "Deep Leakage from Gradients," in *Proc. Advances in Neural Information Processing Systems*, 2019, pp. 1–11.

[11] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov, "See Through Gradients: Image Batch Recovery via GradInversion," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 337–16 346.

[12] Y. Deng, F. Lyu, J. Ren, H. Wu, Y. Zhou, Y. Zhang, and X. Shen, "AUCTION: Automated and Quality-Aware Client Selection Framework for Efficient Federated Learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, pp. 1996–2009, 2022.

[13] J. Xu and H. Wang, "Client Selection and Bandwidth Allocation in Wireless Federated Learning Networks: A Long-Term Perspective," *IEEE Transactions on Wireless Communications*, vol. 20, no. 2, pp. 1188–1200, 2021.

[14] M. Zhang, G. Zhu, S. Wang, J. Jiang, Q. Liao, C. Zhong, and S. Cui, "Communication-efficient federated edge learning via optimal probabilistic device scheduling," *IEEE Transactions on Wireless Communications*, vol. 21, no. 10, pp. 8536–8551, 2022.

[15] N. H. Tran, W. Bao, A. Zomaya, M. N. H. Nguyen, and C. S. Hong, "Federated Learning over Wireless Networks: Optimization Model Design and Analysis," in *Proc. IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1387–1395.

[16] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935–1949, 2020.

[17] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep Learning with Differential Privacy," in *Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 308–318.

[18] K. Wei, J. Li, M. Ding, C. Ma, H. Su, B. Zhang, and H. V. Poor, "User-level privacy-preserving federated learning: Analysis and performance optimization," *IEEE Transactions on Mobile Computing*, vol. 21, no. 9, pp. 3388–3401, 2021.

[19] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. V. Poor, "Federated Learning With Differential Privacy: Algorithms and Performance Analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.

[20] D. Liu and O. Simeone, "Privacy for Free: Wireless Federated Learning via Uncoded Transmission With Adaptive Power Control," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 170–185, 2021.

[21] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[22] B. Zhao, X. Liu, W.-N. Chen, and R. Deng, "CrowdFL: privacy-preserving mobile crowdsensing system via federated learning," *IEEE Transactions on Mobile Computing*, 2022.

[23] A. Triastcyn and B. Faltings, "Federated Generative Privacy," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 50–57, 2020.

[24] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Blockchain and Federated Learning for Privacy-Preserved Data Sharing in Industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4177–4186, 2020.

[25] Y. Qu, L. Gao, T. H. Luan, Y. Xiang, S. Yu, B. Li, and G. Zheng, "Decentralized Privacy Using Blockchain-Enabled Federated Learning in Fog Computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5171–5183, 2020.

[26] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

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

[28] O. Li, J. Sun, X. Yang, W. Gao, H. Zhang, J. Xie, V. Smith, and C. Wang, "Label Leakage and Protection in Two-party Split Learning," in *International Conference on Learning Representations*, 2021.

[29] S. Pal, M. Uniyal, J. Park, P. Vepakomma, R. Raskar, M. Bennis, M. Jeon, and J. Choi, "Server-Side Local Gradient Averaging and Learning Rate Acceleration for Scalable Split Learning," *arXiv preprint arXiv:2112.05929*, 2021.

[30] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8485–8493.

[31] D.-J. Han, H. I. Bhatti, J. Lee, and J. Moon, "Accelerating federated learning with split learning on locally generated losses," in *ICML 2021 Workshop on Federated Learning for User Privacy and Data Confidentiality. ICML Board*, 2021.

[32] S. Oh, J. Park, P. Vepakomma, S. Baek, R. Raskar, M. Bennis, and S.-L. Kim, "LocFedMix-SL: Localize, Federate, and Mix for Improved Scalability, Convergence, and Latency in Split Learning," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 3347–3357.

[33] X. Liu, Y. Deng, and T. Mahmoodi, "Wireless distributed learning: A new hybrid split and federated learning approach," *IEEE Transactions on Wireless Communications*, 2022.

[34] A. Abedi and S. S. Khan, "FedSL: Federated Split Learning on Distributed Sequential Data in Recurrent Neural Networks," *arXiv preprint arXiv:2011.03180*, 2020.

[35] V. Turina, Z. Zhang, F. Esposito, and I. Matta, "Federated or Split? A Performance and Privacy Analysis of Hybrid Split and Federated Learning Architectures," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 250–260.

[36] Y. Tian, Y. Wan, L. Lyu, D. Yao, H. Jin, and L. Sun, "FedBERT: When Federated Learning Meets Pre-Training," *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2022.

[37] D. Romanini, A. J. Hall, P. Papadopoulos, T. Titcombe, A. Ismail, T. Cebere, R. Sandmann, R. Roehm, and M. A. Hoeh, "Pyvertical: A vertical federated learning framework for multi-headed splitnn," *arXiv preprint arXiv:2104.00489*, 2021.

[38] F. Jameel, Z. Hamid, F. Jabeen, S. Zeadally, and M. A. Javed, "A Survey of Device-to-Device Communications: Research Issues and Challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2133–2168, 2018.

[39] Z. He, T. Zhang, and R. B. Lee, "Model inversion attacks against collaborative inference," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 148–162.

[40] T. Wang, Y. Zhang, and R. Jia, "Improving robustness to model inversion attacks via mutual information regularization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 666–11 673.

[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[42] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[44] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[45] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[46] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization," in *Proc. Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 7611–7623.

[47] Y. Zhan, J. Zhang, Z. Hong, L. Wu, P. Li, and S. Guo, "A Survey of Incentive Mechanism Design for Federated Learning," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2021.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3309633

Jinglong Shen received the B.S. degree in Information Engineering from Xidian University, Xi'an, China, in 2021 and is currently pursuing the Ph.D. degree in Information and Communication Engineering with the School of Telecommunications Engineering, Xidian University, Xi'an, China. His research interests include federated learning, wireless communication, and mobile edge computing.

Nan Cheng received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Waterloo in 2016, and B.E. degree and the M.S. degree from the Department of Electronics and Information Engineering, Tongji University, Shanghai, China, in 2009 and 2012, respectively. He worked as a Post-doctoral fellow with the Department of Electrical and Computer Engineering, University of Toronto, from 2017 to 2019. He is currently a professor with State Key Lab. of ISN and with School of Telecommunications Engineering, Xidian University, Shaanxi, China. He has published over 90 journal papers in IEEE Transactions and other top journals. He serves as associate editors for IEEE Transactions on Vehicular Technology, IEEE Open Journal of the Communications Society, and Peer-to-Peer Networking and Applications, and serves/served as guest editors for several journals. His current research focuses on B5G/6G, AI-driven future networks, and space-air-ground integrated network.

Xiucheng Wang received a B.S. degree in telecommunication engineering from Xidian University in 2021, and is currently pursuing an M.S. degree from Xidian University. His research area of interest is digital twin and graph neural networks of the wireless network.

Feng Lyu (IEEE M'18-SM'22) received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2018. During respective Sept. 2018-Dec. 2019 and Oct. 2016-Oct. 2017, he worked as a postdoctoral fellow and was a visiting PhD student in BBCR Group, Department of Electrical and Computer Engineering, University of Waterloo, Canada. He is currently a professor with the School of Computer Science and Engineering, Central South University, Changsha, China. His research interests include mobile networks, beyond 5G networks, big data measurement and application design, and edge computing. He is the recipient of the best paper award of IEEE ICC 2019. He currently serves as associate editor for IEEE Systems Journal and Peer-to-Peer Networking and Applications, and served as TPC members for many international conferences. He is a member of the IEEE Computer Society, Communication Society, and Vehicular Technology Society.

Wenchao Xu is a research assistant professor at The Hong Kong Polytechnic University. He received his Ph.D. degree from University of Waterloo, Canada, in 2018. Before that he received the B.E. and M.E. degrees from Zhejiang University, Hangzhou, China, in 2008 and 2011, respectively. In 2011, he joined Alcatel Lucent Shanghai Bell Co. Ltd., where he was a Software Engineer for telecom virtualization. He has also been an Assistant Professor at School of Computing and Information Sciences in Caritas Institute of Higher Education, Hong Kong. His research interests includes wireless communication, Internet of things, distributed computing and AI enabled networking.

Zhi Liu received the Ph.D. degree in informatics in National Institute of Informatics. He is currently an Associate Professor at The University of Electro-Communications. His research interest includes video network transmission and mobile edge computing. He is now an editorial board member of Springer wireless networks and IEEE Open Journal of the Computer Society. He is a senior member of IEEE.

Khalid Aldubaikhy (Member, IEEE) received the B.E. degree from Qassim University, Buraydah, Saudi Arabia, in 2008, the M.A.Sc. degree in electrical and computer engineering from Dalhousie University, Halifax, NS, Canada, in 2012, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2019. He is currently an Assistant Professor at the Department of Electrical Engineering, Qassim University, Qassim, Saudi Arabia. His research interests include millimeter-wave wireless networks, medium access control, MU-MIMO, and 6G cellular networks.

Xuemin (Sherman) Shen (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990.

He is currently a University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research interests include network resource management, wireless network security, the Internet of Things, 5G and beyond, and vehicular ad hoc and sensor networks. He is a registered Professional Engineer of Ontario, Canada, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, a Chinese Academy of Engineering Foreign Member, and a Distinguished Lecturer of the IEEE Vehicular Technology Society and Communications Society. He received the R.A. Fessenden Award from IEEE, Canada, in 2019, the Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, the James Evans Avant Garde Award from the IEEE Vehicular Technology Society in 2018, the Joseph LoCicero Award in 2015 and Education Award in 2017 from the IEEE Communications Society, and the Technical Recognition Award from Wireless Communications Technical Committee in 2019 and AHSN Technical Committee in 2013. He has also received the Excellent Graduate Supervision Award from the University of Waterloo in 2006 and the Premier's Research Excellence Award (PREA) from the Province of Ontario, Canada, in 2003. He served as the Technical Program Committee Chair/the Co-Chair for IEEE Globecom'16, IEEE Infocom'14, IEEE VTC'10 Fall, and IEEE Globecom'07, and the Chair for the IEEE Communications Society Technical Committee on Wireless Communications. He is the President Elect of the IEEE Communications Society. He was the Vice President for Technical & Educational Activities, the Vice President for Publications, a Member-at-Large on the Board of Governors, the Chair of the Distinguished Lecturer Selection Committee, and a member of IEEE Fellow Selection Committee of the ComSoc. He served as the Editor-in-Chief for the IEEE INTERNET OF THINGS JOURNAL, IEEE Network, and IET Communications.