The Response to Reviewers for Major Revision

IEEE TRANSACTIONS ON MOBILE COMPUTING,

Manuscript ID: TMC-2023-02-0157

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

Dear Editor and Reviewers

Thank you for giving us the opportunity to submit a revised draft of the manuscript "RingSFL: An Adaptive Split Federated Learning Towards Taming Client Heterogeneity" for publication in IEEE Transactions on Mobile Computing. We appreciate the time and effort that editors and the reviewers dedicated to providing feedback on our manuscript and are grateful for the insightful comments and valuable improvements to our paper. We have incorporated most of the suggestions made by the reviewers. Please refer to the followings for a point-by-point response to the reviewers' comments and concerns. All page numbers refer to the revised manuscript file.

Reviewer #1

The Reviewer's Comment #1.1

My main concern with RingSFL is the threat of model inversion attacks. Section 3.6.1 was a good addition to discuss the threat of gradient-based reconstruction attacks. However, if an adversary has some model access, either white-box or black-box, then there is a threat that the feature maps shared during forward propagation could be used to reconstruct the original data [1]. This would especially be a threat when a client's propagation length is 1, meaning an adversary may only need to learn a linear mapping to reconstruct the original data.
I am curious about the authors thoughts on these kinds of attacks, and would encourage them to at least add a subsection to discuss this topic. Under what conditions would RingSFL be vulnerable to such attacks? If the authors feel RingSFL can defend against such attacks under certain circumstances, then it would be nice to have a plot similar to Figure 5 or even experiments such as those shown in Figure 16 showing the success or failure of model inversion attacks.

Response:

Thank you so much for your constructive comments. We have added a special subsection to discuss the privacy issues arising from D2D communication between clients. As discussed in [1], there are three main settings for model inversion attacks (recover training data according to the corresponding feature maps): **white-box setting**, **black-box setting**, and **query-free setting**. We discuss each of these three settings and explore under which settings RingSFL can defend against model inversion attacks, and under which settings it is vulnerable.

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.

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.

[1] 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.

The Reviewer's Comment #1.2

Section 3.5 is a bit confusing to read when comparing with Figure 4. The text states that an overlapping layer is defined as a layer where multiple clients contribute gradients. However, the figure shows an overlapping layer only being present at one client. Do you say a client is contributing gradients to the layer if that client is the one that calculated the loss and started backpropagation? Or am I misunderstanding?

Response:

Thank you so much for your constructive comments. We have redesigned Fig. 4 and explained in detail our definition of overlapping layer.

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

As illustrated in Figure R1, 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 propagation flow of client 0 is illustrated by the blue arrows, which propagate through $\mathcal{W}_{0,(0,1)}^t$ and $\mathcal{W}_{1,(2,5)}^t$, respectively, and the propagation flow of client 1 is illustrated by the red arrows, which propagate through $\mathcal{W}_{1,(0,3)}^t$ and $\mathcal{W}_{0,(4,5)}^t$, respectively. It can be noted that $\mathcal{W}_{1,(2,3)}^t$ is contained by both $\mathcal{W}_{1,(2,5)}^t$ and $\mathcal{W}_{1,(0,3)}^t$, i.e., the propagation flows of both client 0 and client 1 pass over $\mathcal{W}_{1,(2,3)}^t$ (with both blue and red arrows passing over them), so we call $\mathcal{W}_{1,(2,3)}^t$ the overlapping fully connected layers in this RingSFL system. Whereas there is no layer on client 0 that lies within the propagation range of both clients (no layer has both red and blue arrows passing through), so there is no overlapping layer on client 0.

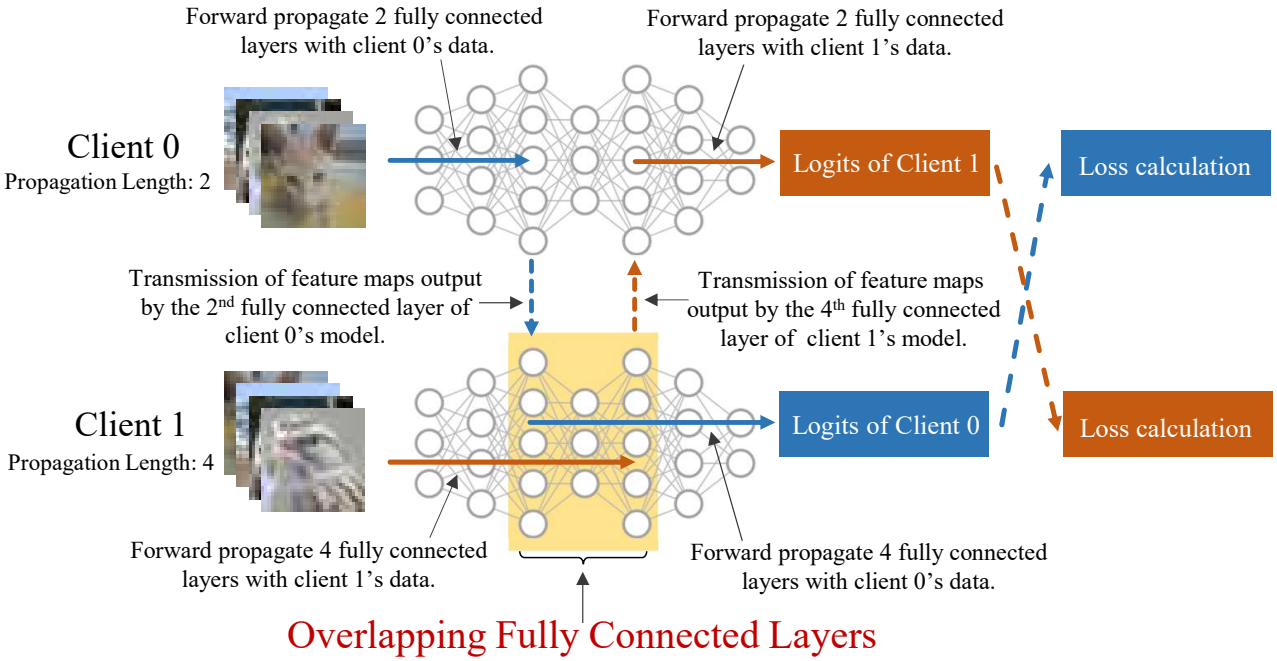


Figure R1: RingSFL training over 2 clients; $L_0 : L_1 = 2 : 4$.

The Reviewer's Comment #1.3

In Equation 7, since $\mathcal{U}$ is a set, it would make more sense to use $|\mathcal{U}|$ to denote the cardinality of the set, the norm notation is a bit confusing.

Response:

Thank you so much for your constructive comments. We have modified equation 7 by using $|\mathcal{U}|$ to denote the cardinality of the set $\mathcal{U}$. Equation 7 is now rewritten as

$$\mathcal{W}^t_{i,(j)} = \mathcal{W}^t_{i,(j)} - \eta|\mathcal{U}_{i,(j)}| \sum_{k\in\mathcal{U}_{i,(j)}} a_k \boldsymbol{g}^t_{k,(j)}. \tag{1}$$

The Reviewer's Comment #1.4

The claim that the eavesdropper does not know the propagation lengths seems strong. If the eavesdropper can access gradients, would it not be able to intercept messages from the server and learn the propagation length for each client?

Response:

Thank you so much for your constructive comments. We remove the claim that the eavesdropper does not know the propagation length in the manuscript and discuss the privacy performance of RingSFL in the presence of a strong capability eavesdropper (one that can eavesdrop on all clients' local model parameters and propagation lengths) in subsection 3.6.1 of the manuscript.

According to existing research [1, 2], 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

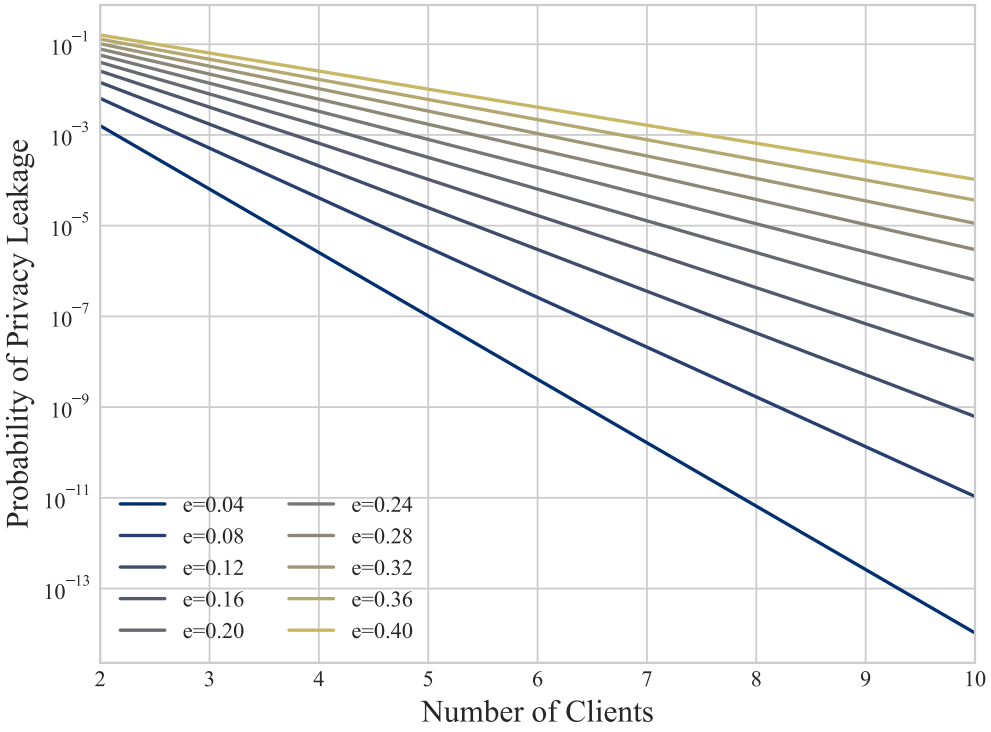


Figure R2: Impact of Number of Clients and Eavesdropping on Privacy Leakage.

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. Denoting by e_i the probability that the communication link between u_i and the server is eavesdropped, the probability of privacy leakage can be expressed as

$$P = \prod_{i=0,\cdots,N-1} e_i. \tag{2}$$

It can be seen from (2) that the privacy leakage probability is influenced by the eavesdropping probability e_i and the number of clients N. Figure R2 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).

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

[2] 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.

The Reviewer's Comment #1.5

Table 3: it would be nice for the reader if the best accuracy for each model and data distribution was in bold.

Response:

Thank you so much for your constructive comments. As shown in Table R1, we modified Table 3 by redrawing it as a three-line table with the best accuracy marked in bold, and the secondary marked in underline.

Table R1: Top-1 Accuracy (%) of Each Model under Different Algorithms.

	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 / $\mathbf{98.84 \pm 0.08}$	98.80 ± 0.14 / $\mathbf{98.86 \pm 0.09}$

The Reviewer's Comment #1.6

Figures 7, 9, and 10: there appears to be a plotting error, some lines from the box extend out to the left.

Response:

Thank you so much for your constructive comments. We are sorry for these plotting errors, it may be caused by the incompatibility of the svg images we uploaded with your pdf reader, we have replaced these images with pdf format and hope it will solve your problem.

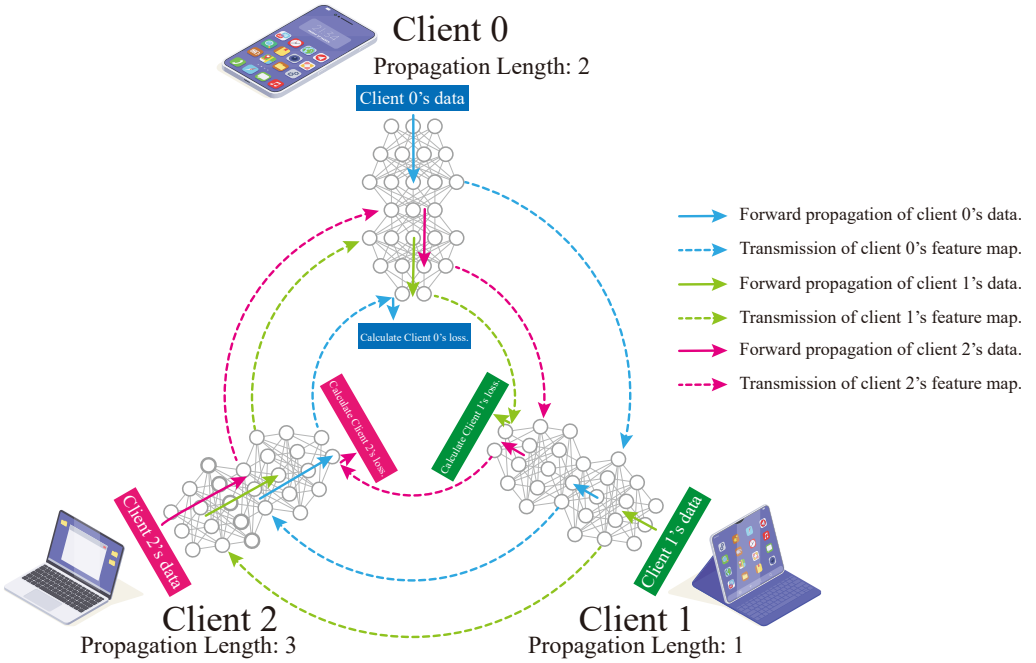
Reviewer #2

The Reviewer's Comment #2.1

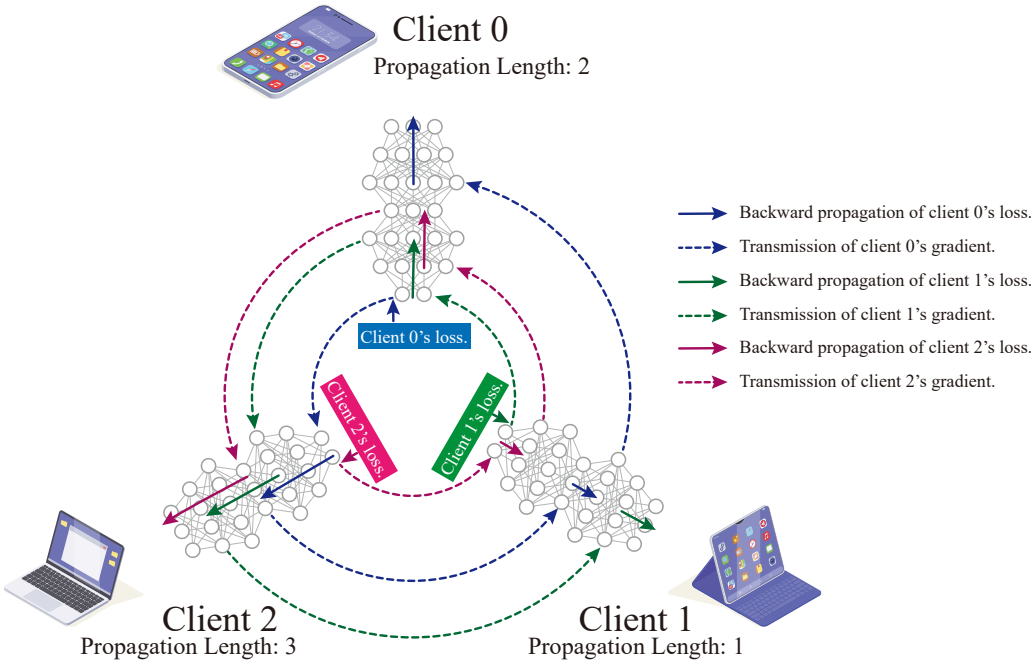
Figure describing the overall algorithm: First of all, a detailed figure showing forward/backward propagation process of RingSL is necessary. Without this, a reader has to understand the detailed algorithm with only equations, which is very complicated. Authors may consider the case with 3 users and provide details on the forward/backward propagation process. This figure may also help the readers to better understand why there are overlapping updates for each layer.

Response:

Thank you so much for your constructive comments. As shown in Figure R3, we have added an image for illustrating the RingSFL forward/backward propagation process.



Forward Propagation



Backward Propagation

Figure R3: Propagation flow of RingSFL with 3 clients. $L_0 : L_1 : L_2 = 2 : 1 : 3$

The Reviewer's Comment #2.2

Fig. 4: Related to the comment above, if I understood the algorithm correctly the example of Fig. 4 has significant errors. In Fig. 4, although the number of (input & output & hidden layers) are 3, the number of fully connected layer is 2. Hence, the case with $L_0 = 1$ and $L_1 = 2$ doesn't make sense because the workload should be defined as the number of fully connected layers, which determines the amount of workload. In Fig. 4, the data of client 0 skips the first fully connected layer and directly becomes the input of the second fully connected layer, which doesn't make sense if the input layer size and the hidden layer size is different. Similarly, the data of client 1 skips the second fully connected layer. Moreover, the concept of "overlapping layer" should also be the "overlapping fully connected layer" (in case of MLP, and "overlapping convolutional layer" in case of CNNs) instead of "overlapping hidden layer". The authors should make all these points clearer and better describe the overall algorithm with more efforts.

Response:

Thank you so much for your constructive comments. We are very sorry for the confusion we caused you. As illustrated in Figure R1, we have redesigned Fig. 4 (Fig. 5 in current version of manuscript) and adjusted our text presentation. In the figure, two clients collaborate to train a multilayer perceptron (MLP) containing 6 fully connected layers, where the propagation length of client 0 is set to $L_0 = 2$ and that of client 1 is set to $L_1 = 4$.

During training, the data (a mini-batch) of client 0 is first forward-propagated in client 0's local model for two layers, and then the feature maps output from the second fully-connected layer is transmitted to client 1, which then feeds the received feature maps into the third fully-connected layer of client 1's local model and forward-propagates it for four layers to obtain the model output logits. The forward propagation process of client 1 is similar, and the propagation process of both clients is parallel.

Since layers 3 and 4 of the local model of client 1 lie within the propagation path of both client 0 and client 1 (with red and blue arrows propagating through), these two layers are overlapping fully connected layers.

The Reviewer's Comment #2.3

Fig. 1: Fig. 1 is also somewhat confusing. What does the colored hidden layers mean in Fig. 1? If these represent the propagation length, it doesn't make sense since the sum of the portion of responsible layers should become 1, which is not the case in Fig. 1.

Response:

Thank you so much for your constructive comments. We are very sorry for the confusion we caused you. As shown in Figure R4, we redesigned Fig. 1, in which we removed the content about neural network training and kept only the system components of RingSFL to illustrate the system architecture of RingSFL. For the training process of RingSFL (including forward/backward propagation) please refer to Figure R3.

The system components of RingSFL are the same as FedAvg, containing a server for neural network aggregation and multiple clients involved in training. There are communication links between the clients and the server to transmit model parameters to the server for aggregation, while the clients form a ring communication topology among themselves to transmit feature maps.

The Reviewer's Comment #2.4

The term split: I am also not sure if the term "split" is appropriately used in many sentences (e.g., By properly splitting the model and allocating it to heterogeneous clients), because all clients have the full model during training, not the split model; each client has different number of propagation layers, but eventually, most of the clients would update the whole model by repeating forward/backward process using data of all clients in the system.

Response:

Thank you so much for your constructive comments. RingSFL does not really split the model, but rather "split" it at the logical level during forward/backward propagation. The term split here refers to assigning different propagation lengths to

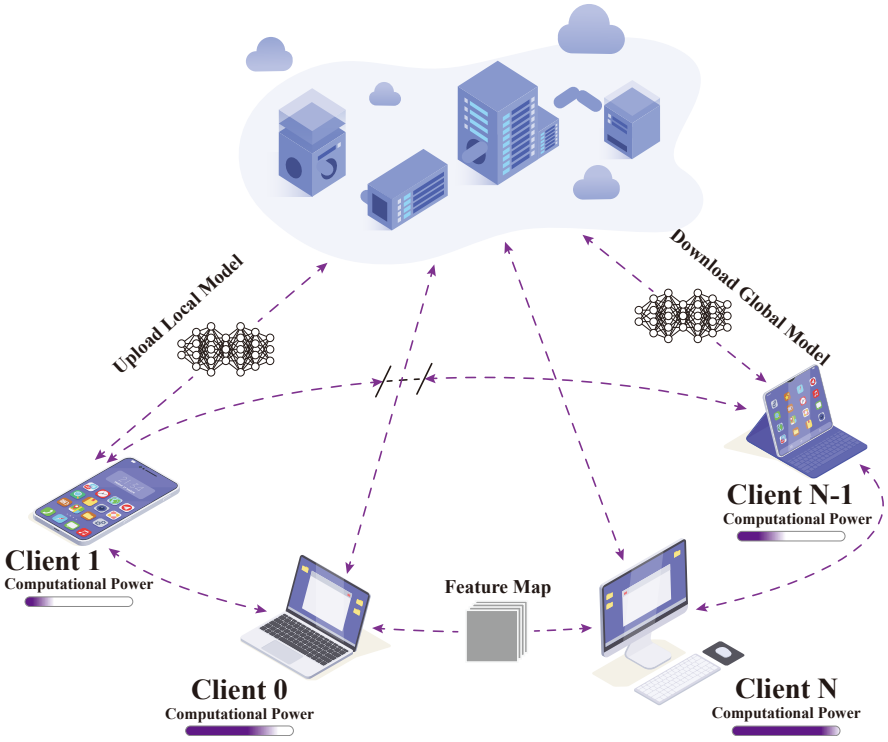


Figure R4: The architecture of RingSFL.

different clients, and does not really mean splitting the model. We thank the reviewer for the valuable comments, and we have improved the corresponding expressions where they are not appropriate.

The Reviewer's Comment #2.5

Privacy for sharing features among clients: Although RingSL has better privacy against the server by sending the blended models, it requires sharing the features to the adjacent clients, which can also cause privacy issues. The authors may need to discuss about this.

Response:

Thank you so much for your constructive comments. We have added a special subsection to discuss the privacy issues associated with the transmission of feature maps between clients in RingSFL. The transmission of feature maps is mainly subject to the risk of model inversion attacks, i.e., recovering the corresponding real original data based on the transmitted feature maps. As discussed in [1], there are three main settings for model inversion attacks: **white-box setting**, **black-box setting**, and **query-free setting**. We discuss the privacy performance of RingSFL under these three settings, respectively.

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.

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.

[1] 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.

The Reviewer's Comment #2.6

Multiple mini-batch: In federated learning, multiple mini-batch updates are performed before sending the updated model to the server to handle the communication bottleneck. However, in this scheme, to perform another mini-batch update, the full communication round among clients are required for the forward/backward propagation process.

Response:

Thank you so much for your constructive comments. RingSFL is similar to FedAvg in that instead of sending every mini-batch update to the server for aggregation, multiple mini-batches of training are also performed before sending the updated model to the server. We have modified our corresponding text and notation to clarify this issue.

The Reviewer's Comment #2.7

Finally, I am not very convinced with the ring-shaped architecture itself for handling the straggler issue. For example, when a specific client becomes outage due to the battery issue as described in the introduction, it results in significant bottleneck since every forward/backward signals cannot pass that client. In contrast, in conventional FL with a single server, it is acceptable to have some stragglers in each round because one can ignore them.

Response:

Thank you so much for your constructive comments. RingSFL has a different idea than conventional FL when it comes to handling stragglers. In conventional FL, as described by the reviewer, the server can handle this problem by simply ignoring the stragglers and dropping their local models. However, this leads to two problems:

- The server usually sets a time threshold, and when a client exceeds this time threshold without sending its local

model to the server, the server will ignore the client. However, in order to ensure that the server does not ignore too many clients, this time threshold is usually set to a large value, which results in the server still needing to wait for a long time when there is a straggler in the system, making the system less efficient in training. However, in RingSFL, by allocating the training load, even stragglers can be trained in a short time and the server will not wait for a long time.

- Another issue that deserves our consideration is whether it is reasonable for the server to directly ignore a particular client? In the edge network environment, there exists an particular case where the client involved in training has a large local dataset but only very poor computational resources. Since it has a large dataset, the local model of that client will have a large weight in the server aggregation, which has a large impact on the final global model performance. If the server frequently ignores the model updates of this client due to its slow update speed, it will have a negative impact on the performance of the final global model. In RingSFL, the server tries its best effort to keep all clients' model updates to avoid this problem.

On the other hand, in RingSFL, the client participating in the training may indeed have power or communication problems, which may cause that client to drop out of the training and become offline in the middle of the training. This is a communication topology construction and maintenance problem, which is also a problem to be solved in our future work. Technically speaking, RingSFL can handle the issue of client offline by the following methods:

- Before each communication round: Before each communication round starts, the server will evaluate the communication and computational capabilities of the clients currently participating in the training, set the communication topology according to the capabilities of each client, and avoid the problematic nodes in advance.
- During each communication round: we assume that the connections between clients and between clients and the server are TCP connections. When a TCP connection is broken, both communicating parties of that TCP connection will receive a link break signal. During training, when a client goes offline, that client's TCP connection to other clients and to the server will be broken and the server will be able to capture a link break signal. When the server finds that a TCP connection is broken, it terminates the current round of training in advance, receives the local models of the clients that are still online for aggregation, and reconfigures the ring communication topology to start the next round of training.

Reviewer #3

The Reviewer's Comment #3.1

Does a ring structure leads to exposure of client identities (as clients communicate with each other directly).

Response:

Thank you so much for your constructive comments. In RingSFL, communication between clients can be divided into two phases: establishing a communication connection phase, and transmitting a feature map phase. When a client establishes a communication connection with other clients, whether the client's identity is exposed depends on the communication protocol it uses. If the communication protocol used by the client is anonymous or does not require authentication, then there will be no issue of exposing the identity of the client. This is a communication protocol design issue, which is not related to neural network training and is out of the scope of this paper, so we mainly discuss whether the client's identity will be exposed when transferring feature maps between clients.

When a client transmits a feature map to other clients over an established communication link, there is mainly risk of model inversion attacks, i.e., other clients may recover the original data of that client based on the received feature maps and use it to determine the identity of that client. Therefore, we add a special subsection to discuss the privacy performance of RingSFL against model inversion attacks. As discussed in [1], there are three main settings for model inversion attacks: **white-box setting**, **black-box setting**, and **query-free setting**. We discuss the privacy performance of RingSFL in 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.

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.

[1] 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.

The Reviewer's Comment #3.2

Could distributing training process based on available lead to privacy problems for machine configuration? (e.g., say that the client is aware of model architecture, but not parameters, could still infer the amount of resources that the remaining clients have based on the propagation length assigned to itself).

Response:

Thank you so much for your constructive comments. In RingSFL, the propagation length of each client is proportional to its own computational resources, so the client can infer the sum of the computational resources of the remaining other clients based on its own propagation length. However, the client cannot infer exactly what the computational resources of each of these clients are, so we do not consider this to be a privacy issue.

The Reviewer's Comment #3.3

The author listed a series of related works from different perspective and then say that RingSFL improves on existing techniques. Could you discuss deeper into how does the approach compare to existing techniques? I noticed that there are "Benefits and limitations" subsections, are these intended for this purpose? If they are then the authors could point out specific design decisions that corresponds to the differences.

Response:

Thank you so much for your constructive comments. We have described in more detail how RingSFL compares with existing techniques and where its advantages are, and further elaborated on the design motivation of the RingSFL training mechanism in the realated work subsection.

The Reviewer's Comment #3.4

How does RingSFL handles failure of client? Does the model get distributed?

Response:

Thank you so much for your constructive comments. In RingSFL, the client participating in the training may have power or communication problems, which may cause that client to drop out of the training and become offline in the middle of the training. This is a communication topology construction and maintenance problem, which is also a problem to be solved in our future work. Technically speaking, RingSFL can handle the issue of client offline by the following methods:

- Before each communication round: Before each communication round starts, the server will evaluate the communication and computational capabilities of the clients currently participating in the training, set the communication topology according to the capabilities of each client, and avoid the problematic nodes in advance.
- During each communication round: we assume that the connections between clients and between clients and the server are TCP connections. When a TCP connection is broken, both communicating parties of that TCP connection will receive a link break signal. During training, when a client goes offline, that client's TCP connection to other clients and to the server will be broken and the server will be able to capture a link break signal. When the server finds that a TCP connection is broken, it terminates the current round of training in advance, receives the local models of the clients that are still online for aggregation, and reconfigures the ring communication topology to start the next round of training.

The Reviewer's Comment #3.5

Overlapping layers — Is it an explicit choice that different components of the network converges differently? I understand the learning rate could be adaptive when overlapping layers occur, but is there a more balanced approach so that the model has the same performance as the non-distributed one.

Response:

Thank you so much for your constructive comments. In RingSFL, the server allocates different propagation lengths to different clients based on their computational resources, so the presence of overlapping layers can easily occur. In the earlier version of RingSFL (RingSFLv1), instead of making the learning rate adaptive, we adopted a more balanced update approach, i.e., all layers of the network used the same learning rate when updating. With this setup, we found that RingSFLv1 performs about the same (in model accuracy) as vanilla FL on the Non-IID dataset and slightly better than vanilla FL on the IID dataset.

However, with our further study, we found that the presence of the overlapping layer can effectively improve the performance of RingSFL (in model accuracy, on both IID and Non-IID datasets). As discussed in the manuscript, there are multiple propagation flows on the overlapping layer, and thus the sum of multiple gradients is subtracted each time the parameters of the overlapping layer are updated, which corresponds to a higher aggregation frequency (aggregation is performed each time the backpropagation is performed) on the overlapping layer. Therefore, its gradient is more capable of driving the parameters of the overlapping layer to the global optimum, which can effectively improve the performance of the global model. In order to better exploit this property of the overlapping layer, we make the learning rate adaptive and thus propose

RingSFLv2. When updating the parameters of the overlapping layer, we use a larger learning rate than the other layers so that it can have a larger step size in the gradient direction and thus drive the whole model to converge to the global optimum.

Our experiments also demonstrate the necessity of making the learning rate adaptive. As shown in Figure R5 and Figure R6, RingSFLv2 achieves the best model performance on both IID and Non-IID datasets, with significant improvement compared to RingSFLv1. In Figure R7 and Figure R8, we performed the overlapping layer ablation experiments, i.e., the number of overlapping layers in the system is adjusted by manually allocating the propagation lengths of each client. When the propagation lengths of each client are equal (2:2:2:2:2), RingSFLv2 is equivalent to vanilla FL, and there is no overlapping layer at this time. When the propagation lengths is set to 6:1:1:1:1, the propagation lengths is allocated most unevenly and the most overlapping layers exist. It can be seen that the model accuracy increases as the number of overlapping layers increases, so we argue that this unbalanced update (overlapping layer has a larger learning rate) is necessary.

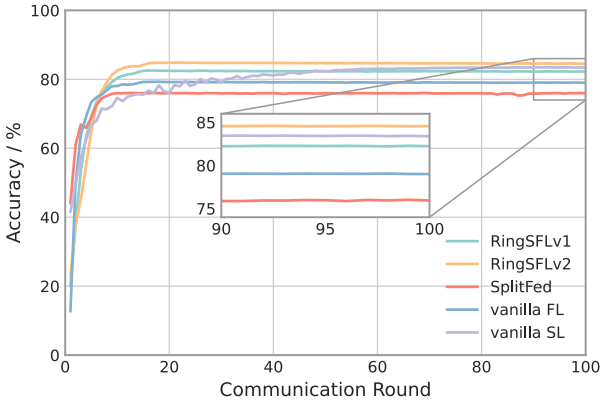


Figure R5: Testing convergence of ResNet18 on CIFAR10 (IID) under different algorithms.

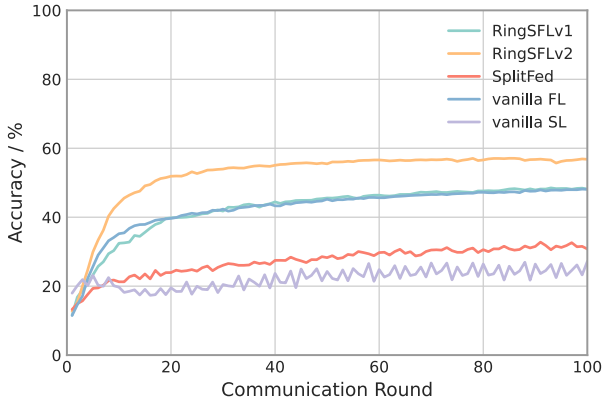


Figure R6: Testing convergence of ResNet18 on CIFAR10 (Non-IID) under different algorithms.

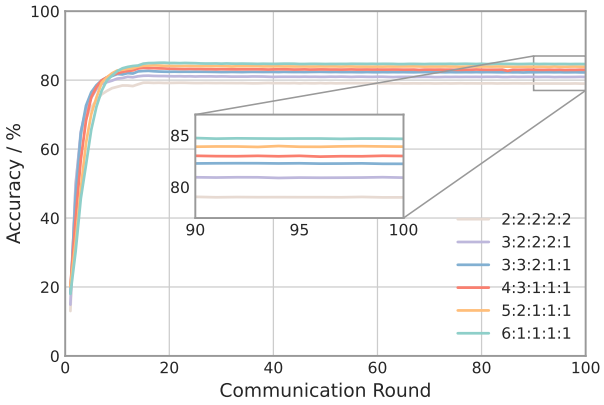


Figure R7: Testing convergence of ResNet18 on CIFAR10 (IID) under different propagation lengths.

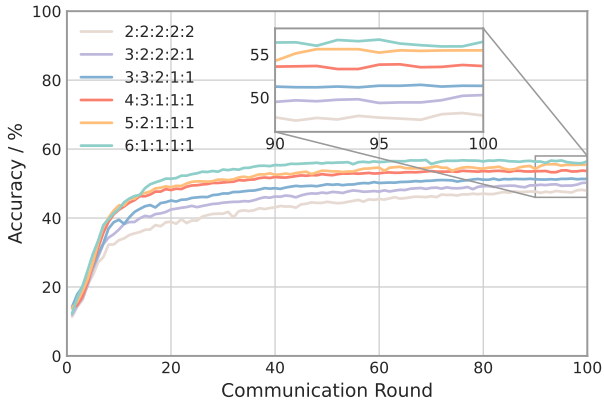


Figure R8: Testing convergence of ResNet18 on CIFAR10 (Non-IID) under different propagation lengths.

The Reviewer's Comment #3.6

Could a client with significant amount of resources be able to infer propagation length of all other clients.

Response:

Thank you so much for your constructive comments. It is possible for a client with significant resources to infer the propagation lenghts of other clients, e.g., a client with significant communication resources can obtain the propagation lenghts of other clients by eavesdropping on the communication links between other clients and the server, and a client with significant computational resources can infer the propagation lenghts of other clients by training additional auxiliary neural networks. Therefore, we remove the claim that the eavesdropper does not know the propagation lenghts in the manuscript, and discuss the privacy performance of RingSFL in the presence of a strong capability eavesdropper (this eavesdropper can

eavesdrop on all clients' local model parameters and propagation lenghts) in Subsection 3.6.1 of the manuscript.

According to existing research [1, 2], 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. Denoting by e_i the probability that the communication link between u_i and the server is eavesdropped, the probability of privacy leakage can be expressed as

$$P = \prod_{i=0,\cdots,N-1} e_i. \tag{3}$$

It can be seen from (3) that the privacy leakage probability is influenced by the eavesdropping probability e_i and the number of clients N. Figure R2 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).

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

[2] 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.