This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems

Privacy-Preserving Computation Offloading for Parallel Deep Neural Networks Training

Yunlong Mao, *Member, IEEE,* Wenbo Hong, *Student Member, IEEE,* Heng Wang, *Student Member, IEEE,* Qun Li, *Fellow, IEEE,* and Sheng Zhong,*Member, IEEE*

Abstract—Deep neural networks (DNNs) have brought significant performance improvements to various real-life applications. However, a DNN training task commonly requires intensive computing resources and a huge data collection, which makes it hard for personal devices to carry out the entire training, especially for mobile devices. The federated learning concept has eased this situation. However, it is still an open problem for individuals to train their own DNN models at an affordable price. In this paper, we propose an alternative DNN training strategy for resource-limited users. With the help of an untrusted server, end users can offload their DNN training tasks to the server in a privacy-preserving manner. To this end, we study the possibility of the separation of a DNN. Then we design a differentially private activation algorithm for end users to ensure the privacy of the offloading after model separation. Furthermore, to meet the rising demand for federated learning, we extend the offloading solution to parallel DNN models training with a secure model weights aggregation scheme for the privacy concern. Experimental results prove the feasibility of computation offloading solutions for DNN models in both solo and parallel modes.

Index Terms—Deep Neural Network, Federated Learning, Computation Offloading, Data Privacy, Model Parallelism

1 Introduction

People with mobile devices such as smartphones, Google glasses, or HoloLens can sense the environment and use collected data (e.g., image and sound) to train a deep neural network (DNN) for various applications. Usually, there are two possible ways for mobile devices to get a well-trained DNN model, sending all private data to a central server that has sufficient resources, or performing distributed training with a local DNN trained on each device. Obviously, the first way is more suitable for mobile devices with limited computing resources. But user's privacy will be violated seriously if the server is untrusted [1]. The second way has a higher requirement for computing resources. Even if it's possible for mobile devices to perform distributed training, user's private data can still be violated by an active adversary [2], [3]. To tackle this problem, we give an alternative solution for privacy-aware DNN training for single client in previous work [4], [5].

To meet the rising demand for parallel DNN model training, we extend the computation offloading solution to a popular parallel DNN training framework, i.e., federated deep learning (FL). FL trains duplicate models on multiple clients in a parallel style. Each FL client trains a DNN model with private data locally and upload the updating result to a central server. Then a globally updated model could be downloaded from the central server after the aggregation. This paradigm requires intensive computation and a stable high-speed network for FL clients. It is quite hard for portable smart devices to join and benefit from this paradigm. Since portable smart devices are becoming data producers, it is necessary to take their practical demands into account. If we can offload part of the training workload to a trusted server while processing private data locally, FL will be more affordable to resource-limited devices. However, assuming a trusted server is impracticable in the real world. Hence, it is crucial to study how to offload parallel model training to an untrusted server safely.

Privacy issue in deep learning has been identified in some recent studies [2], [3], [6]. To tackle this issue, some efficient privacy-aware training mechanisms have been proposed. A differentially private gradient computing solution is given in [7] for the training phase. Privacy-preserving parameter aggregation solutions for distributed learning have been studied in [1], [8]. A differentially private parameters updating solution is introduced in [8] while a secure parameter aggregation solution based on masking technique and threshold secret sharing is proposed in [1]. Targeting at privacy-preserving fine-tuning, [9] migrates the learning process from a client to a server after mixing basic features extracted by clients with noise. However, none of the existing work has focused on DNN computation offloading.

We try to fill the gap by offering alternative solutions for privacy-preserving DNN offloading in both solo and parallel manners. The proposed solutions are based on the observation that layers inside a DNN are loosely coupled. The network can be divided into two parts and deployed separately as long as the intermediate results keep consistent. To deal with the leakage threat of the user's private training data, we propose a differentially private activation

- *Yunlong Mao, Wenbo Hong, Heng Wang and Sheng Zhong are with the State Key Laboratory for Novel Software Technology, the Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China.*
- *Qun Li is with the Department of Computer Science, College of William & Mary, Williamsburg, VA, US*

Part of this work has been published in the proceedings of the third ACM/IEEE Symposium on Edge Computing (SEC 2018) and the USENIX workshop on Hot Topics in Edge Computing (HotEdge 2018).

1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems

algorithm, which helps the user hide sensitive information from an untrusted offloading server. Meanwhile, we study the separation strategy of a DNN model for the user's offloading by modeling the user's computing resource, privacy budget, and desired model quality as an optimizing problem. The optimization result shows that a simple but effective separation strategy is to keep just the first layer of a DNN on the user side and offload the rest part to an untrusted server. That's the main conclusion of our previous work [4], [5].

In this paper, we have extended the privacy-preserving offloading solution for a single DNN model training task to the parallel DNN models training scenario. In particular, we first revisit the previously proposed differentially private activation algorithm and improve its utility for each user through concentrated privacy analyzing method. Then we propose a privacy-preserving offloading solution for users of parallel DNN models training task with this improved activation algorithm. The model weights aggregation is crucial for parallel training. The offloaded parts of all parallel models can be aggregated simply since they are deployed on the same server. But it is difficult to have model weights on the user's side aggregated because private data will be disclosed if the aggregation is not designed for privacy concerns. To tackle this problem, we propose a secure weights aggregation solution for users' non-offloaded parts with the help of a verifiable secret sharing scheme. In this manner, model weights of the non-offloaded parts can be aggregated secretly and partial users' failure during the aggregation can be tolerated. Furthermore, our solution is also resistant to improper secret shares in case of the existence of malicious users. When compared with another secure aggregation solution [1], ours has fewer travel rounds of communication between the server and users and the resistance to corrupt secret shares. Overall, the contributions of our work can be summarized as follows.

- We propose a computation offloading solution for DNN model training tasks in a private manner by designing a privacy-preserving algorithm for activations calculation.
- By tracing privacy loss, computing cost and training accuracy, we give a study of DNN separation strategy. We recommend that keeping the first layer non-offloaded is the best choice for a trade-off between computing resources, privacy loss and model quality.
- We revisit the proposed private activation algorithm and improve its utility by introducing a sharper privacy analysis technique.
- A privacy-preserving offloading solution for parallel DNN models training is proposed with the design of a secure model weights aggregation for users' non-offloaded parts. The proposed solution has fewer communication rounds and the resistance to corrupt secret shares, compared with other secure aggregation solution.

2 Preliminaries

2.1 Deep Convolutional Neural Network

DNNs, such as [10], [11], [12], [13] have similar architectures in common. The most significant feature is the convolutional layers. A convolutional layer is where filters convolve around input volume. Taking VGG-16 as an example, we demonstrate the architecture and data flow of common DNNs in Figure 1. Model weight dimensions and the number of filters in each convolutional layer are also indicated in the figure. We will use "conv" with numbers to refer to some specific convolutional layer hereafter. Please note that the study in this paper can be adapted to other DNNs while we will use VGG-16 network as an example in the rest.

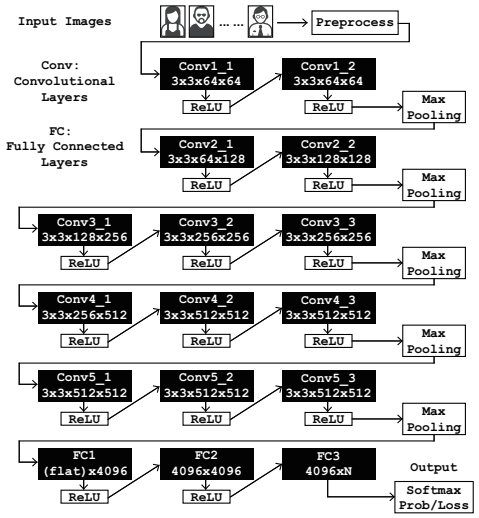


Fig. 1: Network architecture and data flow of VGG-16.

2.2 Differentially Private Mechanism

Differential privacy (DP) [14] provides a promising privacy property. Assuming that all possibly queried datasets of an application compose $\mathcal{D}$, two adjacent datasets D, D' can be defined as two neighbouring datasets differing in a single entry.

Definition 1 (Differential Privacy)**.** *A random mechanism $M : \mathcal{D} \to \mathcal{R}$ satisfies (ϵ, δ)-differential privacy ((ϵ, δ)-DP for short) if for any two adjacent datasets $D, D' \in \mathcal{D}$ and for any subsets of outputs $S \subset \mathcal{R}$ it holds that*

$$P[M(D) \in S] \leq e^{\epsilon} P[M(D') \in S] + \delta. \tag{1}$$

This definition given in [14] allows that original ϵ-differential privacy can be broken with probability δ. A simple way to construct an (ϵ, δ)-DP mechanism for query function $f : \mathcal{D} \to \mathcal{R}$ is adding random perturbation. Gaussian mechanism is a basic mechanism satisfying DP. To construct a Gaussian mechanism having (ϵ, δ)-DP, a noise term should be added to f:

$$M(D) \triangleq f(D) + \mathcal{N}(0, S_f^2 \cdot \sigma^2), \tag{2}$$

where $\mathcal{N}(0, S_f^2 \cdot \sigma^2)$ is a Gaussian distribution with mean 0 and standard deviation $S_f\sigma$, S_f is the sensitivity of f which can be defined as $S_f = \max_{D,D' \in \mathcal{D}} |f(D) - f(D')|$.

2.3 Federated Deep Learning

In a federated deep learning (FL) task, a central server (or parameter server interchangeably) will coordinate n clients for learning jointly. Assume that each client $P_i, i \in [1, n]$ holds a single dataset D_i for training. Before the training

1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems

begins, the central server broadcasts DNN architecture, learning hyperparameters and training task to the clients. When the FL training begins, client P_i, $i \in [1, n]$ will train a local model with D_i individually.

The local training of each client is basically the same as usual. Generally, it is assumed that a mini-batch stochastic gradient descent (SGD) algorithm is used to minimize the loss $\mathcal{L}(\boldsymbol{\theta}^i)$ on P_i's model weights (or parameters interchangeably) $\boldsymbol{\theta}^i$ iteratively. In each iteration, P_i randomly samples training data to form a mini batch $\boldsymbol{d}_i = \{d_1, d_2, \ldots, d_m\}$. Then P_i computes an average loss over the mini-batch as $\frac{1}{m}\sum_{j=1}^{m}\mathcal{L}(\boldsymbol{\theta}^i, d_j)$. Then the gradient $\nabla_{\boldsymbol{\theta}^i}\mathcal{L}(\boldsymbol{\theta}^i)$ can be estimated as

$$\boldsymbol{g}^i = 1/m \sum_{j=1}^{m} \nabla_{\boldsymbol{\theta}^i}\mathcal{L}(\boldsymbol{\theta}^i, d_j). \tag{3}$$

A global training iteration counter $t \in [1, T]$ for coordination is maintained by the central server. In the end of t-th iteration, local model weights should be updated as $\boldsymbol{\theta}_t^i = \boldsymbol{\theta}_{t-1}^i - \eta \boldsymbol{g}^i$, where η is a predefined learning rate. Each P_i, $i \in [1, n]$ is supposed to upload $\boldsymbol{\theta}_t^i$ to the central server. Once the updated weights of all clients have been collected, the central server will perform aggregation for updating following a given strategy. Generally, we will use an averaging strategy for weights aggregation, which is simple and effective. In this way, the central server gives globally updated weights $\bar{\boldsymbol{\theta}}_t = 1/n \sum_{i=1}^{n} \boldsymbol{\theta}_t^i$

At the beginning of the $(t+1)$-th iteration, all clients should download the latest global weights $\bar{\boldsymbol{\theta}}_t$ from the central server and have their local models synced. Then the above procedure will be repeated until the global model has converged or the maximum iteration T is reached.

2.4 DNN Offloading Model

We borrow the basic idea of computation offloading from [15], [16]. In a client-server offloading mode, a cloud (or edge) offloading service provider should keep stable communication with a client (can be mobile or not). In the FL mode, the central server can perform as the offloading service provider while each FL client shares the same communication model as FL setting. Taking the solo client-server training mode as an example, we will give an offloading computation model for DNN training.

First, we should partition a DNN model into two parts $\boldsymbol{\theta}_{server}$ and $\boldsymbol{\theta}_{client}$ for offloading requirement. For the local computation, the client with training dataset D should compute $\boldsymbol{a} = f(\boldsymbol{\theta}_{client}, \boldsymbol{d})$, $\boldsymbol{d} \subset D$ in the forward passing and compute $\boldsymbol{g}_{client} = \frac{1}{m}\sum_{j=1}^{m}\nabla_{\boldsymbol{\theta}_{client}}\mathcal{L}(\boldsymbol{\theta}_{client}, \boldsymbol{d})$ in the backward passing. For the offloading computation, the server should take $\boldsymbol{a}$ as its input in the forward passing and compute $\mathcal{L}(\boldsymbol{\theta}_{server}, \boldsymbol{a})$. In the backward passing, the server should compute $\boldsymbol{g}_{server} = \frac{1}{m}\sum_{j=1}^{m}\nabla_{\boldsymbol{\theta}_{server}}\mathcal{L}(\boldsymbol{\theta}_{server}, \boldsymbol{a})$. Finally, the server and the client can update partitioned weights separately.

2.5 Threat Model

2.5.1 Computation Offloading for Single DNN Training

The server who undertakes offloaded training task is a honest-but-curious, which means that the server is curious about client's private data but will follow the predefined protocols strictly. However, the adversarial server can perform other computation to infer client's privacy as long as it does not interfere with the offloaded computation. We assume that the server has unlimited computing resource. The data privacy of the client can be defined as that the adversary cannot tell whether a specific sample is from the client's dataset or not if the adversarial server has no prior knowledge. The honest-but-curious server has been studied widely in previous work. However, it is the first time to study this threat model in an offloading case.

2.5.2 Computation Offloading for Parallel DNN Training

For parallel DNN training, we assume that each client has established a secure channel with the central server. All clients can also reach each other in secure communication channels, which mean that network attacks like eavesdropping are excluded. Any FL client is possible to drop out occasionally during the parallel training. Since multiple clients are considered in parallel training case, the threat model will be complex than the single DNN training case. The adversarial server shares the same assumptions with the single DNN training case.

Generally, threats may come from the server, other clients or both. We say a client is honest-but-curious if this client aims to disclose some other client's private data by observing the parallel training while following the designed protocol normally. We say a client is compromised if this client is controlled by and share views with the adversarial server. The compromised client is assumed to be no more than honest-but-curious. It is assumed that the amount of semi-honest clients and compromised clients should be no more than half of the total clients. Furthermore, we take somewhat malicious clients into account. A somewhat malicious client can use corrupt messages to disturb the whole training procedure, which has not been considered in existed work like [1]. Meanwhile, it is noted that data manipulation attacks like poisoning attack against the FL is out of our discussion and need to be studied separately. In summary, the security requirements that need to be ensured are as follows.

- (R0) If the amount of honest clients is no less than a predefined threshold, the parallel training is resistant to clients drop out and somewhat malicious clients.
- (R1) Honest-but-curious server cannot obtain more information than the client aggregation result and necessarily auxiliary intermediate results.
- (R2) Honest-but-curious clients cannot obtain more information than the client aggregation result and meaningless secret shares received from other clients.
- (R3) Honest-but-curious server and its compromised clients cannot get more information than the client aggregation result, necessarily auxiliary intermediate results and meaningless secret shares received from other clients.

3 Related Work

Recently, severe attacks against DNNs have been identified [2], [3], [6], [17]. To address these privacy issues, many efforts have been made. A differentially private DNN training solution for parallel model training is first studied in

1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems

[8]. Within the same training scenario, a secure aggregation scheme for parallel DNN models is proposed in [1]. The authors use masking technique and threshold secret sharing method rather than differential privacy to achieve secure aggregation of model weights. In [18], the authors reconstruct operators of DNN with cryptographic primitives and leveled homomorphic encryption is used to protect private datasets and intermediate results. Secure multi-party computation is a promising solution for general privacy-aware DNNs. But this general solution may suffer from heavy computation and complex communication protocols. It is not suitable for resources-limited devices.

On the other side, server aided DNN training solutions for mobile users have been studied recently. A privacy-preserving deep learning framework with aided cloud server is provided in [19], which describes how to infer with private data while the training phase is done with public data and artificial noise. Another similar study is [9]. The authors proposed a privacy-preserving DNN training solution based on transfer learning. Low-level features are first extracted locally. Then these features are perturbed and sent to the server. Solution has good performance on fine-tuning applications. But it is not proper for training from scratch. The most significant difference between existed studies and ours is that we preserve more precise features for both training and inference. Meanwhile, users in our solution need less resources consumption when compared with related work [1], [20]. Furthermore, we give an offloading solution in private manner for parallel model training, which is not supported by existed work like [9], [19].

Furthermore, the concept of FL is evolving with the development of many research topics, like blockchain. Weng *et.al* propose a federated learning framework, Deepchain [21], which exploits blockchain-based incentive mechanism to incentive participants to behave honestly in parameter updating. Dishonest participants will be detected and punished. This is achieved by verifying the transactions in Deepchain and denying the access of service if one has insufficient values. Pokhrel replaces the global central server in FL with a multi-level blockchain with reputation based incentive mechanism [22]. In this way, the transmission delay of local model result is reduced and the reliability of each local participant is improved. Lu *et.al* report in [23] when combined with blockchain, the model learned by FL is more reliable and more robust. Moreover, as Qu *et.al* mentioned in [24], replacing the central server in FL with blockchain not only ensures decentralized privacy protection but also avoids single point failure.

4 Privacy-Preserving Offloading for Single DNN Training

In DNNs, each hidden layer can be seen as a separate unit taking the output of previous layer as its input. If a client wants to offload the whole DNN training task to the server, privacy issue must be addressed locally. It is not recommended to use high-level artificial noise to perturb private data directly because the utility of data will be damaged seriously [19], [25]. To tackle this problem, we propose to partition the original DNN into two parts and offload a part instead of the whole DNN to the server. Intuitively, this idea is feasible because layers inside DNNs are loosely coupled. Once we select a specific position to separate the whole network like the case shown in Figure 2, the client can hide raw training data and all intermediate results from the untrusted server. All information that the server needs to know for the offloaded computation is the output of the client's part. In this manner, the client can preserve his data privacy with the help of privacy preserving techniques locally.

An important part of this idea is to find out the best position to partition the network. A good partitioning position should achieve high training accuracy and client's data privacy while avoiding large resource consumption at the same time. A main result of our study shows that for a specific DNN, if all output activations of the i-th layer are (ϵ, δ)-DP then weights updating procedure will be ϵ_i-DP for each iteration, where ϵ_i is total privacy budget for the i-th layer. Then we have $\epsilon_i > \epsilon_j$, if $i < j$ for any two convolutional layers. This means that the latter (a latter layer refers to the layer closer to the model output layer) layer we interfere with using a fixed noise, the more privacy is kept but the less model quality we obtain.

For simplicity, we will introduce our offloading solution with a simple partitioning strategy where the client holds the first convolutional layer (with ReLU attached) and feeds the output to the server, who will succeeds the client to finish the following computation in a predefined DNN. Then we will discuss the optimal partitioning position specifically. An offloading example of VGG-16 network is illustrated in Figure 2. The client sends its output activations of the first convolutional layer instead of raw training data to the server. Artificial noises will be added to the output activations for preventing the adversarial server from disclosing client's privacy by reversing the uploaded activations.

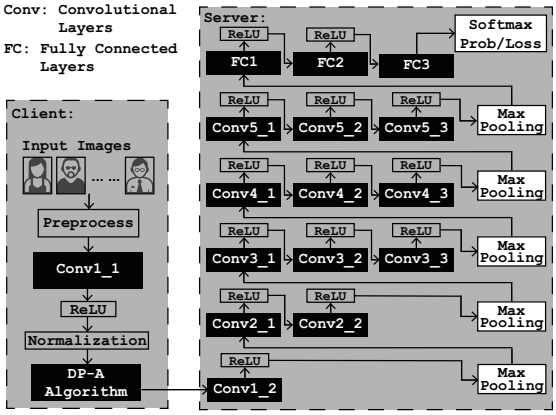


Fig. 2: A VGG-16 network is separated for offloading.

4.1 Differentially Private Activations

As shown in Figure 2, when we partition VGG-16 network into two parts, except for the first convolutional layer, all other layers are offloaded to the server unaltered. When the client communicates with the server, the server part can be seen as a black-box function in client's view and vice versa. In the first convolutional layer, we assume that there are m filters with size $l \times l \times r$, where r indicates the number of color channels. When a data sample d is fed into activating function $f^i, i \in [1, m]$, an activation is

1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems

generated corresponding to a spatial position, denoted by (x, y), $x \in [1, w]$, $y \in [1, h]$, on the surface of sample d. Then function f^i (f will be used equivalently if no ambiguity is caused) can be defined as

$$f^i(d(x,y)) = a^i_{(x,y)} / (\gamma + \alpha \sum_{j=\max(0, i-\frac{u}{2})}^{\min(m-1, i+\frac{u}{2})} (a^j_{(x,y)})^2)^\beta, \quad (4)$$

where u, α, β, γ are constants for normalization, $a^i_{(x,y)}$ is the activation generated by i-th filter at position (x, y).

To protect client's data privacy, we need to ensure output activations of function f for every single data sample is privacy-preserved. Function f can be seen as a specific query on client's datasets D. To construct a (ϵ, δ)-DP application for f with Gaussian mechanism, sensitivity of f on adjacent datasets D, D' should be clarified. Based on the definition of function sensitivity and ReLU's output activation $a^i_{(x,y)} \geq 0$, we can define f's sensitivity S_f as

$$\begin{aligned} S_f = \max_{D, D' \in \mathcal{D}} \big| & a^i_{(x,y)}(D) / (\gamma + \alpha \sum_{j=\max(0, i-\frac{u}{2})}^{\min(m-1, i+\frac{u}{2})} (a^j_{(x,y)}(D))^2)^\beta \\ & - a^i_{(x,y)}(D') / (\gamma + \alpha \sum_{j=\max(0, i-\frac{u}{2})}^{\min(m-1, i+\frac{u}{2})} (a^j_{(x,y)}(D'))^2)^\beta \big| \\ \leq \max_{D \in \mathcal{D}} & a^i_{(x,y)}(D) / (\gamma + \alpha \sum_{j=\max(0, i-\frac{u}{2})}^{\min(m-1, i+\frac{u}{2})} (a^j_{(x,y)}(D))^2)^\beta. \end{aligned}$$

Given sensitivity S_f, when f is applied at a spatial position (x, y) on some data sample, we can use $c^i_{(x,y)} + \mathcal{N}(0, S_f^2 \sigma^2)$ to replace $a^i_{(x,y)}$ as output of f. In this way, we can construct a (ϵ, δ)-DP mechanism for activations (DP-A algorithm for short). Since $0 \leq a^i_{(x,y)} < 1, \forall d \in D$ after pre-processing, we can have a loose bound $0 \leq S_f < 1/\sqrt{2}$ when we select parameter $u = 5, \alpha = 1, \beta = 0.5, \gamma = 2$.

4.2 Privacy-Preserving Weights Updating

After DP activations are transmitted to the server, client's forward passing is finished. To continue the training task, the server should run subsequent training process from the second convolutional layer with activations received from the client as the input. The output of each convolutional layer on the server side can be seen as the composition of DP activations. Then the total loss of network prediction can also be seen as a composited mechanism of multiple DP mechanisms. But it is challenging to calculate a precise privacy loss of this composited mechanism rather than a simple composition.

When multiple DP mechanisms are combined, the privacy loss normally increases because of potentially repeated queries on the same data. If we directly follow the adaptive composition theory, the prediction mechanism should be $(\epsilon', pq\delta + \delta')$-DP, where p, q are dimensions of activations for each filter, $\epsilon' = \epsilon\sqrt{2pqln(\frac{1}{\delta'})} + pq\epsilon(e^\epsilon - 1), \delta' > 0$. However, we find that in the offloaded DNN training, we can actually achieve a tighter bound than the simple composition theory.

Generally, a loss function $\mathcal{L}$ is used to score the DNN model prediction. As to the gradient computation in backward passing, a mini-batch SGD method will be used. If we want to update weights $\boldsymbol{\theta}$ in t-th iteration, we can use $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta * \boldsymbol{g}_t$, where $\boldsymbol{g}_t$ is an averaged estimation across the mini-batch to the gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$, η is a predefined learning rate. And this estimation $\boldsymbol{g}_t$ can be calculated by $\boldsymbol{g}_t = \frac{1}{m} \sum_{i \in [1,m]} \frac{\partial \mathcal{L}(d_i)}{\partial \boldsymbol{\theta}}$, where $\{d_1, d_2, \ldots, d_m\}$ is a mini-batch randomly generated from dataset D. $\mathcal{L}$ regarding d_i is $\mathcal{L}(d_i) = -\log(\frac{e^{o_i}}{\sum_{j=1}^{N} e^{o_j}})$, where o_i is the prediction score of image sample for one label indexed by i among all N labels.

Assuming that we are looking at the first iteration of the training task, all weights in convolutional layers are initialized by sampling from normal distribution $\mathcal{N}(0, 0.01)$. We define a prediction mechanism $M_p : \mathcal{D}^m \to \mathcal{R}$. When M_p applies to two adjacent datasets $D, D' \in \mathcal{D}$, we can calculate the probabilistic differential loss and then prove that M_p satisfies differential privacy[1].

Corollary 1. *If output activations of the first convolutional layer are all (ϵ, δ)-DP, Loss function $\mathcal{L}$ satisfies ϵ_1-DP, where $\epsilon_1 = p \times q \times \epsilon - c$ and c is a small constant.*

Corollary 2. *If output activations of the first convolutional layer are all (ϵ, δ)-DP, weights updating mechanism has $(O(p_b \epsilon_0 \sqrt{T}), \delta_0)$-DP for T iterations, where $\epsilon_0 = c' + c$, $e^{c'} \in (0, 1)$, p_b is the sampling ratio of each batch.*

4.3 Selection of the Partitioning Position

We have some helpful observations about privacy property when we partition the first convolutional layer from the whole network. We now investigate how to achieve the optimal selection of the partitioning position while taking privacy loss, computing resource and training accuracy into consideration. Assume that we will partition the network part before the $(i+1)$-th convolutional layer of VGG-16. This means that layers with indicator less or equal to i will be deployed on the client side, while the rest part will be deployed on the server side. The client should add artificial noise corresponding to the function sensitivity of the i-th convolutional layer to the output of the client part.

Since the output activations of a latter convolutional layer contains more complex compositions of the earlier layers, the output activations of a latter convolutional layer are more sensitive than the earlier ones. Based on the facts above, we can measure the privacy loss of total loss function when the DNN is partitioned before the $(i+1)$-th layer.

Corollary 3. *For a DNN, if all output activations of the i-th layer are (ϵ, δ)-DP then the weights updating mechanism will be ϵ_i-DP for each iteration such that $\epsilon_i < \epsilon_j$, if $i > j$ for any two convolutional layers in this DNN.*

This corollary directly reveals the correlation between privacy loss and partitioning position. However, training accuracy of a DNN model is difficult to be modeled because it involves too many undetermined factors during training. But we can still give an empirical formula to predict training accuracy based on experimental results (please refer to the results given in Evaluation). With unlimited computing resource, training accuracy can be modeled as $1 - \alpha_a / (e^\epsilon + \beta_a)$, where α_a and β_a are empirical parameters. Taking our evaluation results as the example, α_a and β_a are fitted to be 2 and -1.

How much computing resource to be occupied mainly depends on the amount of model weights. As for VGG-16,

1. The proofs of corollaries given in the rest are omitted due to page limit, which can be found in our previous work [4], [5].

1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems

weight quantity for each layer can be found in Figure 1. Basically, a latter convolutional layer will get more weights than a earlier one. It can be seen as a linearly increasing tendency approximately. The computing resource needed for client can be depicted by the quantity of model weights, which will be $\sum_{j=1}^{i} o_j Q(j)$, where $Q(j)$ outputs a normalized quantity of weights of the j-th layer, o_j is 1 if the j-th layer is trainable and 0 if not. Given all these constraints, we can now have our object function to minimize the cost for a client offload a single DNN model training as

$$\begin{aligned} \min_{\epsilon_i} \quad & w_1\epsilon_i + w_2\sum_{j=1}^{i} o_j Q(j) - w_3(1-\frac{\alpha_a}{e^{\epsilon_i}+\beta_a}), \\ \text{s. t.} \quad & w_1, w_2, w_3 > 0, i \in [1, 15], \end{aligned}$$

where w_1, w_2, w_3 are parameters for constraints. Generally, we can assign them equally since we treat these constraints equally. The convexity of the cost function can be easily evaluated by commercial optimization tools. It can be proved that the cost function will get its minimum value when $i = 1$ in this case. This means that partitioning the first convolutional layer from VGG-16 is the best choice for the client.

5 Improved Private Activation Algorithms

During the developing of DP concept, a series of improved analysis methods has emerged, trying to extend the adaptability of DP or to lower the bound of privacy loss. Within amount of these studies, concentrated differential privacy (CDP) and Rényi differential privacy (RDP) are two outstanding alternatives. In particular, we will discuss zero-concentrated differential privacy (zCDP for short, a reformulation of CDP) given by Bun and Steinke and RDP given by Mironov here.

5.1 Zero-Concentrated Differential Privacy

The definition of zCDP given in [26] provides a relaxed reformulation of original CDP. We briefly review the definition of zCDP and some necessary propositions here.

Definition 2 ((ξ, ρ)-zCDP). *A randomized mechanism $M : \mathcal{D} \to \mathcal{R}$ is (ξ, ρ)-zero-concentrated differentially private ((ξ, ρ)-zCDP for short), if for any adjacent datasets $D, D' \in \mathcal{D}$ and all $\alpha \in (1, \infty)$, it holds that $D_\alpha(f(D)||f(D')) \leq \xi$, where D_α is the Rényi divergence of order α between two probability distributions.*

Especially, it is defined that ρ-zCDP is $(0, \rho)$-zCDP. Along with the definition, a concrete construction of zCDP Gaussian mechanism is also given in [26]. If $f : \mathcal{D} \to \mathcal{R}$ is a query on D with sensitivity Δ, then the mechanism $M : \mathcal{D} \to \mathcal{R}$ that yields a sample from $\mathcal{N}(f(D), \sigma^2)$ will satisfy $(\Delta^2/2\sigma^2)$-zCDP. Given zCDP Gaussian mechanism, we can now improve original DP-A with zCDP. Recall that activation function f and its corresponding sensitivity S_f. Perturbed activation $d^i_{(x,y)}$ should be replaced with $\mathcal{N}(c^i_{(x,y)}, \sigma^2)$, where $c^i_{(x,y)}$ is the normalized output of activation function f. According to the propositions given in [26], if M provides ρ-zCDP, then M is $(\rho + 2\sqrt{\rho \log(1/\delta)}, \delta)$-DP, for any $\delta > 0$. Hence, DP-A with zCDP (zCDP-A for short) should have $(S_f^2/2\sigma^2 + 2\sqrt{S_f^2/2\sigma^2 \log(1/\delta)}, \delta)$-DP.

5.2 Rényi Differential Privacy

RDP is another sharp privacy analysis method of original DP, which is built directly upon the definition of Rényi divergence. Rényi divergence provides a divergence measurement between two probability distributions over any possible order. We review the RDP definition given in [27].

Definition 3 ((α, ϵ)-RDP). *A randomized mechanism $M : \mathcal{D} \to \mathcal{R}$ is said to have ϵ-Rényi differential privacy of order α ((α, ϵ)-RDP for short), if for any adjacent datasets $D, D' \in \mathcal{D}$ it holds that $D_\alpha(f(D)||f(D')) \leq \epsilon$, where D_α is the Rényi divergence of order α between two probability distributions.*

With the help of Rényi divergence, RDP is capable of relaxing the privacy loss bounded by the original DP. According to the definition of RDP, an ϵ-DP mechanism is equivalent to a (∞, ϵ)-RDP, where (∞, ϵ) implies (α, ϵ)-RDP for all finite α. To compare with DP-A and zCDP-A, a DP-A with RDP (RDP-A for short) will be constructed by using RDP Gaussian mechanism provided in [27]. If $f : \mathcal{D} \to \mathcal{R}$ is a query on D with sensitivity Δ, then the mechanism $M : \mathcal{D} \to \mathcal{R}$ that yields $f(D) + \mathcal{N}(0, \sigma^2)$ satisfies $(\alpha, \alpha\Delta^2/(2\sigma^2))$-RDP. Given activation function f and its corresponding sensitivity S_f, perturbed activation $d^i_{(x,y)}$ obtained by using RDP has the same formula with DP-A, i.e., $c^i_{(x,y)} + \mathcal{N}(0, \sigma^2)$, where $c^i_{(x,y)}$ is the normalized output of activation function f. The activations perturbed by RDP-A satisfies $(\alpha S_f^2/(2\sigma^2) + \frac{\log(1/\delta)}{\alpha-1}, \delta)$-DP, for any $\delta \in (0, 1)$.

5.3 Comparative Analysis of DP Activation Algorithms

Theoretically, improved privacy-preserving mechanisms with CDP and RDP should have lower privacy loss when compared with original DP. However, it is not clear whether zCDP-A and RDP-A would provide more data utility than DP-A when they share a fixed privacy budget. To be comparable, we construct DP-A, zCDP-A and RDP-A all with Gaussian mechanism using different DP concepts respectively. For each activation result, DP-A provides (ϵ, δ)-DP when noise samples are drawn from $\mathcal{N}(0, \sigma^2)$, where $\sigma > \sqrt{2ln(1.25/\delta)}\frac{S_f}{\epsilon}$ according to the definition of DP-A. Assuming that with the same σ, zCDP-A provides $(S_f^2/2\sigma^2 + 2\sqrt{S_f^2/2\sigma^2 \log(1/\delta)}, \delta)$-DP while RDP-A provides $(\alpha S_f^2/(2\sigma^2) + \frac{\log(1/\delta)}{\alpha-1}, \delta)$ for each activation. In particular, when $\alpha = 2$, $S_f = 1/\sqrt{2}$, $\delta = 0.0001$, DP-A yields $(3.1/\sigma, 10^{-4})$-DP, zCDP-A yields $(1/(4\sigma^2) + 2/\sigma, 10^{-4})$-DP and RDP-A yields $(1/(2\sigma^2) + 4, 10^{-4})$-DP.

A detailed comparison of privacy provided by three activation algorithms are given in Figure 3. Please note that the curves shown here are based on theoretic results, experimental results on real-life data may vary. It can be concluded from the comparison that zCDP-A algorithm provides the lowest privacy loss in the offloaded DNN training application when we select $\sigma > 0.24$ for Gaussian noise sampling. Meanwhile, DP-A algorithm gives the lowest privacy loss when we use $\sigma < 0.23$[2]. But data privacy cannot be protected when σ is less than 0.23 because the additional noise is negligible. Hence, we recommend to use

2. The actual critical point is a fraction between 0.23 and 0.24.

1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems

zCDP-A to replace DP-A for the improvement of privacy guarantee.

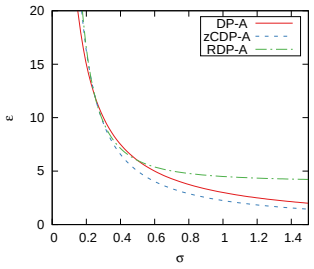


Fig. 3: Privacy provided by different activation algorithms.

6 Privacy-Preserving Offloading for Parallel DNNs Training

In the previous work, we give a privacy-preserving computation offloading solution for a single DNN training task. To meet the rising demand of collaborative learning applications, such as FL, we will provide a privacy-preserving computation offloading solution for parallel DNN models training. Furthermore, we find it possible to reduce the total computing resource consumption of parallel training through privately offloading and model aggregation.

6.1 Offload Parallel DNNs Training with zCDP-A

FL is a widely studied instance of parallel training concepts. But many open problems should be addressed before its further developing [28]. Privacy and resource consumption are two major concerns. By offloading the training task of each client privately to the central server, we provide an alternative solution of FL, which addresses privacy issue and resource issue at the same time. As shown in Figure 4, a privacy-preserving offloaded FL task can be constructed on the basis of our private activation algorithm. Each client can achieve local privacy by using zCDP-A before offloading. The central server should maintain multiple partitioned DNN instances for different clients. Generally, we assume that DNN instances of all clients are partitioned in the same position.

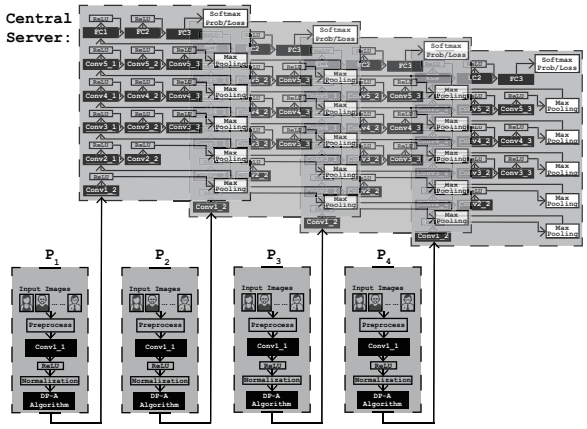


Fig. 4: Clients offload parallel training tasks.

Since all model instances have the same partitioning position, it is possible to improve the efficiency of parallel model training by aggregating the models deployed on the server side. The central server shown in Figure 4 maintains multiple isolated models for all FL clients. If the clients are within the same FL task, their model weights should be synchronized to be identical. Hence, the only difference between parallel models is the input. Benefiting from parallel computing paradigm of DNN, different input activations can be handled as an expanded batch input after the aggregation of models. Taking the case of Figure 4 as an example, if the outputs of DP-A algorithms of four clients are $\boldsymbol{c}^1$, $\boldsymbol{c}^2$, $\boldsymbol{c}^3$ and $\boldsymbol{c}^4$, then the batch input of the aggregated model should be $\{\boldsymbol{c}^1, \boldsymbol{c}^2, \boldsymbol{c}^3, \boldsymbol{c}^4\}$. The gradients of the aggregated model can be calculated by averaging gradients of all samples inside the batch as $\boldsymbol{g} = \frac{1}{m}\sum_{j=1}^{4} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{c}^j)$.

6.2 Secure Aggregation of Non-Offloaded Parts

It is secure to aggregate the parallel models on the server side because the aggregation is equivalent to treating input activations collected from all clients as a newly formed batch. But the tricky part is the parallel models on clients. We have evaluated the offloading solution for parallel DNN training without the aggregation of DNN model deployed on client side (client aggregation for short). Four clients join the FL task aiming at training a CIFAR-10 classification model with noise added to the clients' output which is sampled from $\mathcal{N}(0, 0.1)$. Other settings are the same as introduced in Evaluation. From the evaluation results shown in Table 1, we can conclude that if the non-offloaded parts are not aggregated during the parallel training, global model quality will be frustrated significantly, when compared with the original FL baseline.

TABLE 1: Parallel training without client aggregation.

Non-Offloaded Part	Conv1_1	Conv1_1–2_2	Conv1_1–5_3
Training Accuracy	0.99	0.99	0.99
Test Accuracy	0.66	0.64	0.61

Non-offloaded parts contain knowledge extracted from the raw training data. Publishing model weights of non-offloaded parts directly will put at risk the clients' data privacy. Hence, a secure client aggregation solution is needed. Recently, some practical solutions for secure aggregation of FL have been proposed, like [1], [29]. However, secure client aggregation solution for offloaded parallel training case is still missing. Inspired by previous studies, we propose a hybrid solution for secure client aggregation by combining the DP mechanism with cryptographic tools.

Different from [1], we combine a verifiable secret sharing (VSS) scheme [30] rather than the ordinary Shamir secret sharing with pairwise masking method to provide the mentioned security requirements. Now we need to define some cryptographic primitives first.

- *RSA.enc*, *RSA.dec*: RSA encryption primitives, encrypting transmitter's message with receiver's public key while decrypting transmitter's cipher with receiver's secret key respectively.
- *KA.agree*: Key agreement primitive giving a shared key as its output. In particular, the Diffie-Hellman key agreement will be used.
- *C.commit*, *C.verify*: Commitment primitives, publishing client's commitment to a secret while verifying a published commitment respectively.

1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems

TABLE 2: Comparison between our secure client aggregation solution and related work [1].

	Computation		Communication		Storage		Communication Rounds	Resistance to Corrupt Shares
	client	server	client	server	client	server		
[1]	$O(n^2+mn)$	$O(mn^2)$	$O(n+m)$	$O(n^2+mn)$	$O(n+m)$	$O(n^2+m)$	5	✗
Ours	$O(n^2+(\gamma m+1)n)$	$O(\gamma mn^2)$	$O(n+\gamma m)$	$O((\gamma m+k)n)$	$O(n+\gamma m)$	$O(n^2+m)$	3	✓

- *SS.share*, *SS.reconstruct*: A (k,n)-threshold secret sharing scheme primitives, *SS.share* generates n shares of a secret while *SS.reconstruct* can reconstruct the secret from at least k shares.
- *PRG.gen*: Pseudo-random number generator.

To cover client's model weights, a straightforward method is using randomly generated masks. To recover the masked weights during client aggregation, a pairwise mask technique should be used. To save communication cost, the random seeds for generating pairwise masks are the secrets to be shared among clients. In case of clients' dropping out or failure, the central server can ask other active clients to recover necessary seeds and generated missing masks for the aggregation. To deal with the attack of a active server considered in [1] and the corrupt shares, we construct the solution on the basis of the VSS scheme given in [30]. We now give main steps of secure client aggregation as follows.

(S0) Setup all cryptographic primitives and secure communication channels for the server and clients. Model weights on P_i are denoted by $\boldsymbol{\theta}_{P_i}$. For each client, let p and q denote large primes such that q divides $p-1$ and $n=pq$. Construct key pair pk_i and sk_i with p,q,n. Let g and h denote two elements of G_q. A commitment to some $r\in\mathbb{Z}_q$ can be constructed by $C.commit(r,t)=g^r h^t$, where $t\in\mathbb{Z}_q$ is chosen randomly. To share a secret r with (k,n)-threshold secrete sharing, let $F_i\in\mathbb{Z}_q$ denote the polynomial coefficients of degree at $k-1$ satisfying $F(0)=r$. For masking, we use ordinary addition operator $(+)$ to denote bitwise XOR operation $(\oplus)$.

(S1) P_i chooses a random seed $r_i\in\mathbb{Z}_q$, publishes a commitment C_i to r_i with a randomly chosen $t_i\in\mathbb{Z}_q$: $C_i \leftarrow C.commit(r_i, t_i)$, chooses $F_{io}\in\mathbb{Z}_q$ of degree at $k-1$ and generates n shares of r_i: $\{r_{ij}|j\in[1,n]\}\leftarrow SS.share(r_i)$ and sends $e_{ij}\leftarrow RSA.enc(r_{ij}, pk_j)$ to P_j, generates a seed for generating pairwise masks for client P_j by $s_{ij}\leftarrow KA.agree(g^{r_i}, pk_j)$, generates pairwise masks $m_{ij}\leftarrow PRG.gen(r_{ij})$, $i,j\in[1,n]$, $i\neq j$. P_i chooses $G_{io}\in\mathbb{Z}_q$ randomly and commits to F_{io} using G_{io} and broadcasts $C_{io}\leftarrow C.commit(F_{io},G_{io})$, generates n shares of t_i using G_{io}: $\{t_{ij}|j\in[1,n]\}\leftarrow SS.share(t_i)$ and sends encrypted share $RSA.enc(t_{ij}, pk_j)$ to P_j, $1\le o<k$.

(S2) After receiving encrypted messages from other client P_j, $j\neq i$, P_i decrypts e_{ji} to get $r_{ji}\leftarrow RSA.dec(e_{ji}, sk_i)$, generates seed $s_{ji}\leftarrow KA.agree(g^{r_i}, pk_j)$ and pairwise masks $m_{ij}\leftarrow PRG.gen(s_{ji})$. Then P_i decrypts the cipher, gets t_{ji}, verifies the correctness of r_{ji} by $C.verify(r_{ji}, t_{ji}, \prod_{1\le o<k} C.commit(C_{jo}^{i^o}))$. If the commitment verification passes then continues, otherwise aborts. Then P_i sends masked weight $\boldsymbol{\theta}'_{P_j}\leftarrow\boldsymbol{\theta}_{P_j}+\sum_{i\neq j} m_{ij}$ to the server.

(S3) The server checks the received weights $\boldsymbol{\theta}'_{P_i}$ and yields the dropping out clients set O. For client P_o in O, the server recovers the rand $r_o\leftarrow SS.reconstruct(\{r_{oj}|j\in[1,n], j\neq o\}$ and the seed $s_{oj}\leftarrow KA.agree(g^{r_i}, pk_j)$. Then the server performs client aggregation and broadcasts the result $\boldsymbol{\theta}_{client}\leftarrow 1/(n-|O|)\sum_{P_i\notin O}\boldsymbol{\theta}'_{P_i}+\sum_{P_o\in O}\sum_{j\neq O} PRG.gen(s_{oj})$.

7 Analysis and Comparison

7.1 Security and Communication

According to [30], the adversary cannot open such a commitment unless he can find $log_g(h)$, which is nearly impossible. But the verification of this commitment is quite easy. By putting RSA encryption, commitment scheme and secret sharing together, we have constructed a VSS in the previous subsection. It is oblivious that the constructed VSS satisfies the requirements given in [30]. Hence, the security of our solution can be proved by following the results of [30] directly.

FL clients may drop out occasionally. Generally, we assume that the clients may drop out before sending intermediate results to the server, after sending intermediate results to the server but before sending shares to other clients, or after sending shares to other clients. In the first case, the parallel training procedure will not be hurt since these clients cannot affect the training at all. In the second case, the server can simply drop the duplicated model weights on the server side of these disconnected clients to ensure the consistency of model aggregation for both sides. For the last case, the server can recover model weights of these disconnected clients by the proposed secure aggregation solution. Since the non-offloaded weights of each client are shared among all the other clients secretly, the model weights of these disconnected clients can be secretly recovered via a (k,n)-threshold VSS scheme as long as the amount of remaining clients is no less than k.

Computation complexity. The computation of each client can be broken down to four parts. Encryption, decryption and key agreement between each pair of clients. Commitments of the random seed and the polynomial coefficients. Verification for the shares of the pairwise seed. Mask generation for client's model weights. If we denote the ratio of client model weights to the total weights amount by $\gamma=\frac{|\boldsymbol{\theta}_{P_i}|}{|\boldsymbol{\theta}|}$, then the total computation complexity of each client is about $(4n+2n^2+2k+\gamma mn)$, which is $O(n^2+(\gamma m+1)n)$. The computation of the server mainly happens in (S3). In the worst case, the server needs to recover all pairwise seed from secret sharing for the aggregation of each weight. Hence the total computation of the server is $O(\gamma mn^2)$.

Communication complexity. The communication of each client can also be broken down to four parts. Public key exchanging in the setup (S0) phase. Sending and receiving encrypted shares and commitments. Sending masked model weights to the server. Sending shares to the server in (S3). Overall, the communication complexity of each client is $O(n+\gamma m)$. The communication of the server is mainly receiving shares and masked weights from all clients, which is $O((\gamma m+k)n)$ in the worst case.

Storage complexity. The clients need to store shared secrets and keys of other clients. The commitments sent and received are not stored because they are used for temporal verification. Adding up these and model weights, the storage complexity of each client is $O(n+\gamma m)$. Meanwhile, the

1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems

server needs to store the model weights and all shares of all clients in the worst case, which is $O(n^2 + m)$.

Compared with related secure weights aggregation work [1], our solution can defend against malicious clients who try to sabotage the aggregation by using corrupt shares, which has not been considered in [1]. Furthermore, our solution has less communication rounds since we use a non-interactive VSS scheme. We summarize the complete comparison between our solution and [1] in Table 2.

7.2 Comparison with the Original FL

The original FL concept is proposed by Google in [31], [32]. The privacy-preserving parallel learning solution proposed in our work has made several modifications to the original FL. Generally speaking, there are three main differences. We cut off the original training flow at a partitioning point of each participant's local model and offload the remaining workload to a central server. Then the artificial noise is added to the output of each participant to ensure differential privacy guarantee. And last but not least, an efficient secure aggregation scheme of non-offloaded parts is introduced in this extension work. We give a clear comparison in Algorithm 1 by identifying our modifications to the original FL algorithm[3] explicitly. Please note that we use a sketchy pseudocode to show main differences. Detailed procedures of zCDP-A and secure aggregation are omitted. The main modifications that we make to the original FL are highlighted in red color.

Algorithm 1 Comparison with the original FL

Require: Pre-processed training data D, The participants indexed by P_i, $i \in [1, n]$, VSS threshold k, maximal training iteration T, learning rate η.

Server procedure:
initialize $\boldsymbol{\theta}_{server}$
for all $t \in [1, T]$ **do**
  $P' \leftarrow$ (random set of at least k participants from P)
  for all $P_i \in P'$ **do**
    $\boldsymbol{d}^i \leftarrow P_i$ procedure
    $\boldsymbol{\theta}^i_{server} \leftarrow \boldsymbol{\theta}^i_{server} - \eta \nabla \mathcal{L}(\boldsymbol{\theta}^i_{server}, \boldsymbol{d}^i)$
  end for
  $\boldsymbol{\theta}_{server} \leftarrow \sum_{i \in [1,n]} \frac{n_i}{n} \boldsymbol{\theta}^i_{server}$
end for

Client P_i procedure:
initialize $\boldsymbol{\theta}^i_{client}$
for all batch $b \in D^i$ **do**
  $\boldsymbol{\theta}^i_{client} \leftarrow \boldsymbol{\theta}^i_{client} - \eta \nabla \mathcal{L}(\boldsymbol{\theta}^i_{client}, b)$
end for
$\boldsymbol{d}^i \leftarrow$ zCDP-A output regarding $\boldsymbol{\theta}^i_{client}$
$\boldsymbol{\theta}^i_{client} \leftarrow$ secure aggregation of P'
return $\boldsymbol{d}^i$ to the server

3. The FL algorithms introduced in [31], [32] have distinct differences due to different concerns, such as communication efficiency. We rewrite the original FL by summarizing basic procedures shared by algorithms given in [31], [32] and following the *FederatedAveraging* algorithm in [32] if any ambiguity happens.

8 Evaluation

We have implemented privacy-preserving offloaded DNN training solutions for single model and parallel models. The experimental results are obtained with VGG-16 network. Three real-life datasets are used for evaluation, Labeled Face in the Wild dataset (LFW), CIFAR-10 dataset and SVHN dataset. All convolutional layers in VGG-16 have $stride = 1, pad = 1$. Max pooling size is 2×2. Mini-batch size is 64. Momentum coefficient is 0.9. Learning rate is initialized with 0.01 and exponentially decayed with factor 0.1. Besides, for the local normalization unit that we use in the partitioning layer, $u = 5, \alpha = 1, \beta = 0.5, \gamma = 2$. Without any further statement, we assume that the first layer is partitioned from VGG-16 while the rest part is offloaded.

8.1 Offloaded Single Model Training

To evaluate the feasibility of offloading DNN training with DP-A algorithm, we record the loss and accuracy of the model and give the results in Figure 5. The baseline is a local training result of VGG-16 without any privacy protection. Other three figures are offloaded training results with different epsilon values. It is obvious that small epsilon makes training process more unstable. When $\epsilon = 2$–5, DP-A based offloading solution can achieve both strong privacy and high accuracy.

To perform finetuning on a pre-trained VGG-16 model, weights in fully connected layers will be tuned while the other weights keep frozen. Tuning result in the same setting with no noise added is seen as the baseline. Results of tuning with different epsilons are shown in Figure 6. High accuracy can be achieved in early learning stage. But adding noises to activations can affect learning speed. When $\epsilon = 2$–5, we can still get high accuracy and strong privacy after slightly more epochs. When $\epsilon = 3$, it takes no more than 5 epochs for offloaded tuning to achieve similar accuracy with the baseline.

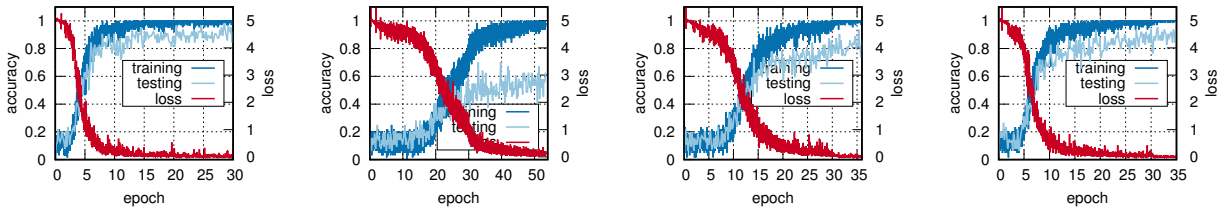
To investigate how partitioning position would affect offloaded training, we perform offloaded training tasks with different partitioning positions. The same level noises are added to client's output, which is equivalent to $\epsilon = 5$ in the first convolutional layer. We choose four partitioning positions in the network uniformly. As shown in Figure 7, the latter the partitioning position is, the harder the training process is. When the network is partitioned before "conv5_1" layer, activations are so sensitive that the training cannot proceed normally.

8.2 Offloaded Parallel Model Training

LFW dataset is seriously unbalanced which is not suitable for parallel model training like FL. Hence, we use another two real-life datasets, i.e., CIFAR-10 and SVHN which are more suitable for FL, to evaluate the offloaded parallel model training. We use a central server to coordinate four clients for parallel learning. The FL protocol is basically the same with [32]. To show the effect of partitioning position to offloaded parallel model training, we choose two different positions to partition the VGG-16. By "conv1_1", we mean that only the first layer is partitioned from the rest layers.

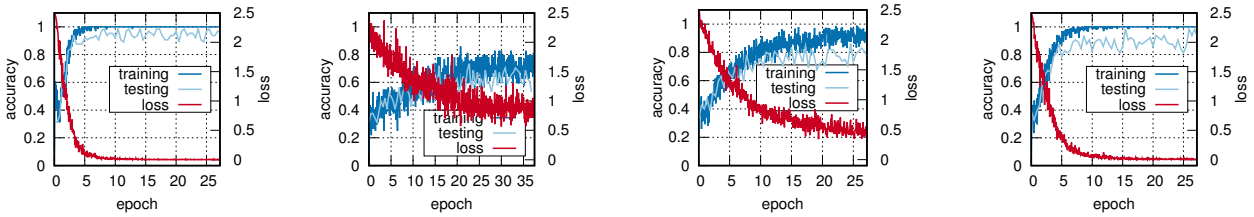
1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems



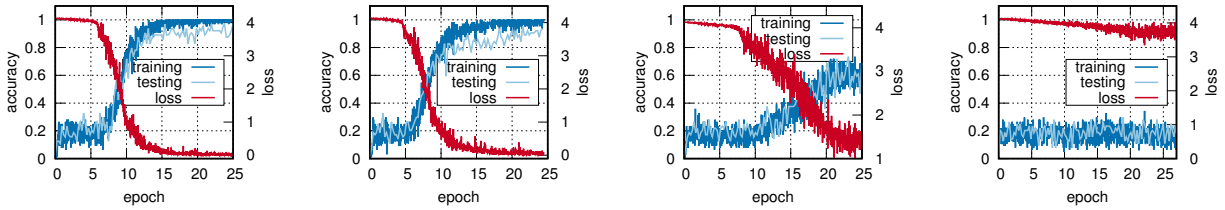
baseline $\epsilon = 1$ $\epsilon = 2$ $\epsilon = 5$

Fig. 5: Offloaded single DNN model training with DP-A algorithm on FLW dataset.



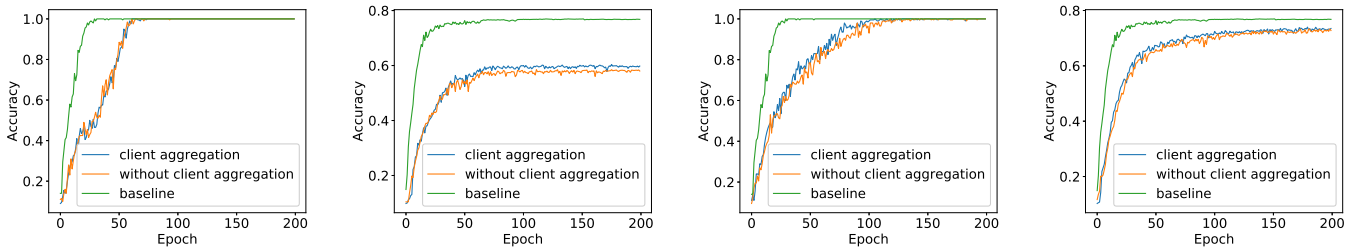
baseline $\epsilon = 1$ $\epsilon = 2$ $\epsilon = 5$

Fig. 6: Offloaded Finetuning for single DNN model with DP-A algorithm on FLW dataset.



"conv1_1" "conv2_1" "conv4_1" "conv5_1"

Fig. 7: Offloaded training results on FLW dataset with different partitioning positions.



training, "conv1_1" test, "conv1_1" training, "conv2_2" test, "conv2_2"

Fig. 8: Offloaded parallel model training results on CIFAR-10 dataset.

By "conv2_2", we mean that the layers from "conv1_1" to "conv2_2" (included) are partitioned from the rest and deployed on client side.

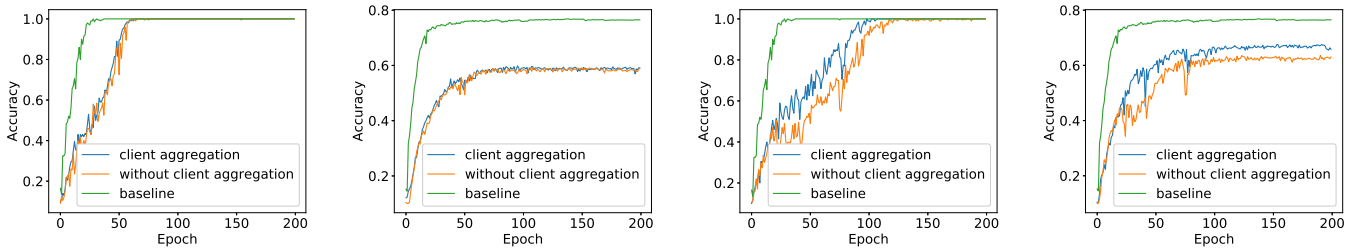
Offloaded parallel training results on CIFAR-10 and SVHN datasets are shown in Figure 8 and Figure 9 respectively. The baseline result is obtained from a ordinary FL task with the same setting but no computation offloading. The same level noises are added to each client's output, which is equivalent to $\epsilon = 2$ for the first convolutional layer. First, we can easily conclude that client aggregation can improve parallel training results significantly for both datasets. Second, we can find that overfitting happens in all training tasks. But the overfitting of 'conv1_1" cases is more serious than "conv2_2" cases. We believe that the output of "conv2_2" is more sensitive to the fixed level noise which eases the overfitting by coincidence. This observation is consistent with our result of partitioning position and the study of preventing the overfitting with random noises [33].

To investigate the effect of the privacy budget, we evaluate the offloaded parallel training solution (with secure client aggregation) with different privacy budgets (ϵ) on both CIFAR-10 and SVHN datasets. The evaluation result is shown in Figure 10. When $\epsilon = 5$, our offloading solution can achieve an acceptable result. But when $\epsilon \leq 1$, the parallel training results will be frustrated. This is basically consistent with our previous study of the single DNN model training.

Since secure client aggregation uses several cryptographic tools, the overhead of the solution is important for evaluation. Hence, we evaluate our secure client aggregation solution separately and give the averaged result of each component across 100 run times in Figure 11. The evaluation is performed in the same experimental environment as the client. The X-axis indicates the threshold k of the secret sharing scheme while the Y-axis indicates data size (bits) and running time (microseconds). The total size of shares and commitments will increase linearly along with the share amount. Generating each share of a secret is around 50 microseconds. Generating each commitment will take about 300 microseconds. Verifying the commitment is very efficient and takes about 12 microseconds for each share. The

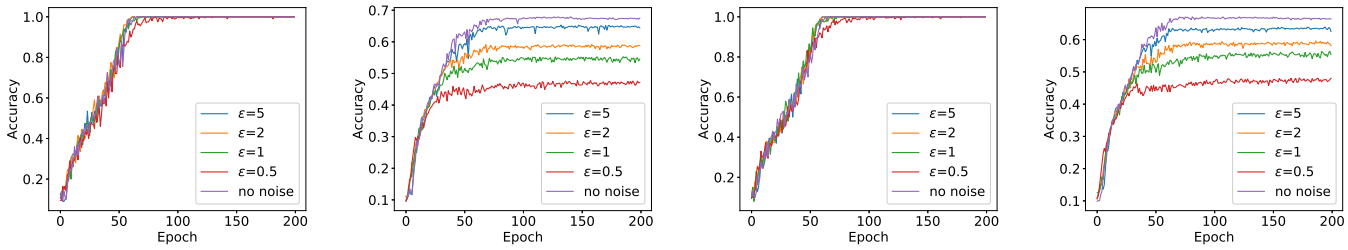
1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems



training, "conv1_1" test, "conv1_1" training, "conv2_2" test, "conv2_2"

Fig. 9: Offloaded parallel model training results on SVHN dataset.



training, CIFAR-10 test, CIFAR-10 training, SVHN test, SVHN

Fig. 10: Offloaded parallel model training with different epsilons on CIFAR-10 and SVHN datasets.

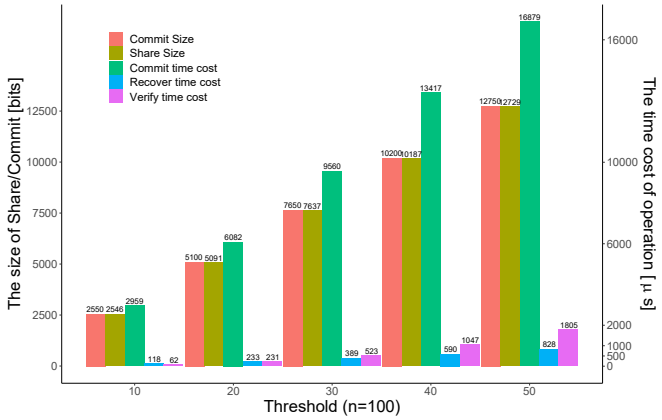


Fig. 11: Evaluation results of secure client aggregation.

time cost of generating and verifying commitments also increase linearly along with the share amount. But recovering the secret requires that running time increases exponentially with the secret amount. As reported in related work [1], the total running time of each client is about 300ms when the number of clients is 100 while our secure aggregation solution takes about 45ms in no drop-out setting.

9 Conclusion

This paper provides privacy-preserving computation offloading solutions for DNN model training tasks in solo mode and parallel mode respectively. To secure the user's data privacy, we design a DP activations algorithm that provides privacy guarantee in the activation level. In the extension, we address a client aggregation problem by employing a non-interactive VSS scheme and masking technique which provides fewer communication rounds and more security guarantees. We evaluate two proposed solutions on real-life datasets. The results show that our proposed solutions have acceptable model quality while preserving a small privacy budget and limited resource consumption.

However, the proposed parallel offloading solution has limitations. When we offload clients' workload to the central server, we assume that the server has unlimited computing resources. This assumption is reasonable when we compare the resource of a commercial corporation with individuals. But a company or a service provider also has its own running budget. How to balance the resource consumption between the server and clients will be studied in our future work.

Acknowledgement

The authors would like to thank the reviewers for the time and efforts they have kindly made on this paper. This work was supported in part by National Key R&D Program of China (2018YFB1004301), BK20190294, NSFC-61902176, NSFC-61872176, the Fundamental Research Funds for the Central Universities No. 14380069, US National Science Foundation grant CNS-1816399. This work was also supported in part by the Commonwealth Cyber Initiative.

References

[1] 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 *Proceedings of the 2017 ACM SIGSAC CCS*, 2017, pp. 1175–1191.

[2] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: Information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC CCS*, 2017, pp. 603–618.

[3] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 691–706.

[4] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "A privacy-preserving deep learning approach for face recognition with edge computing," in *Proc. USENIX Workshop Hot Topics Edge Comput.(HotEdge)*, 2018, pp. 1–6.

[5] ——, "Learning from differentially private neural activations with edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 90–102.

[6] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 3–18.

[7] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC CCS*, 2016, pp. 308–318.

1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2020.3040734, IEEE Transactions on Parallel and Distributed Systems

[8] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22Nd ACM SIGSAC CCS*, 2015, pp. 1310–1321.

[9] S. A. Osia, A. S. Shamsabadi, S. Sajadmanesh, A. Taheri, K. Katevas, H. R. Rabiee, N. D. Lane, and H. Haddadi, "A hybrid deep learning architecture for privacy-preserving mobile analytics," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4505–4518, 2020.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[11] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *IEEE CVPR*, 2014, pp. 1701–1708.

[12] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *Proceedings of the British Machine Vision Conference*, 2015, pp. 41.1–41.12.

[13] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE CVPR*, 2015, pp. 815–823.

[14] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, pp. 211–407, 2014.

[15] W. Liu, J. Cao, L. Yang, L. Xu, X. Qiu, and J. Li, "Appbooster: Boosting the performance of interactive mobile applications with computation offloading and parameter tuning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 6, pp. 1593–1606, 2017.

[16] M. Hu, Z. Xie, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Heterogeneous edge offloading with incomplete information: A minority game approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2139–2154, 2020.

[17] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22Nd ACM SIGSAC CCS*, 2015, pp. 1322–1333.

[18] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, 2016, pp. 201–210.

[19] J. Wang, J. Zhang, W. Bao, X. Zhu, B. Cao, and P. S. Yu, "Not just privacy: Improving performance of private deep learning in mobile cloud," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2407–2416.

[20] J. Zhang, Y. Zhao, J. Wang, and B. Chen, "Fedmec: Improving efficiency of differentially private federated learning via mobile edge computing," *Mobile Networks and Applications*, pp. 1–13, 2020.

[21] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang, and W. Luo, "Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[22] S. R. Pokhrel, "Towards efficient and reliable federated learning using blockchain for autonomous vehicles," *Computer Networks*, p. 107431, 2020.

[23] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4298–4311, 2020.

[24] 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*, 2020.

[25] N. Mohammed, R. Chen, B. C. Fung, and P. S. Yu, "Differentially private data release for data mining," in *ACM KDD*, 2011, pp. 493–501.

[26] M. Bun and T. Steinke, "Concentrated differential privacy: Simplifications, extensions, and lower bounds," in *Theory of Cryptography Conference*. Springer, 2016, pp. 635–658.

[27] I. Mironov, "Rényi differential privacy," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017, pp. 263–275.

[28] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[29] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 1–11.

[30] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991, pp. 129–140.

[31] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

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

[33] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, "Robustness of classifiers: from adversarial to random noise," in *Advances in Neural Information Processing Systems*, 2016, pp. 1632–1640.

Yunlong Mao received the B.S. and Ph.D. degrees in computer science from Nanjing University in 2013 and 2018, respectively. He is currently an assistant researcher with the Department of Computer Science and Technology in Nanjing University. His current research interests include security, privacy and machine learning.

Wenbo Hong is a graduate student in the Department of Computer Science at Nanjing University. His research interests include data privacy and deep learning.

Heng Wang is a graduate student in the Department of Computer Science at Nanjing University. His research interests include security, machine learning and blockchain.

Qun Li received the PhD degree from Dartmouth College. His recent research focuses on wireless, mobile, and embedded systems, including pervasive computing, smart phones, energy efficiency, smart grid, smart health, cognitive radio, wireless LANs, mobile ad-hoc networks, sensor networks, and RFID systems. He is a fellow of the IEEE.

Sheng Zhong received the B.S. and M.S. degrees from Nanjing University in 1996 and 1999, respectively, and the Ph.D. degree from Yale University in 2004, all in computer science. He is interested in security, privacy, and economic incentives.

1045-9219 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: William & Mary. Downloaded on February 10,2021 at 23:29:35 UTC from IEEE Xplore. Restrictions apply.