Crowdlearning: Crowded Deep Learning with Data Privacy

Linlin Chen*, Taeho Jung†, Haohua Du*, Jianwei Qian*, Jiahui Hou*, Xiang-Yang Li‡
*Illinois Institute of Technology, †University of Notre Dame, ‡University of Science and Technology of China
{lchen96, hdu4, jqian15, jhou11}@hawk.iit.edu, tjung@nd.edu, xiangyangli@ustc.edu.cn

***Abstract*— Deep Learning has shown promising performance in a variety of pattern recognition tasks owning to large quantities of training data and complex structures of neural networks. However conventional deep neural network (DNN) training involves centrally collecting and storing the training data, and then centrally training the neural network, which raises much privacy concerns for the data producers. In this paper, we study how to enable deep learning without disclosing individual data to the DNN trainer. We analyze the risks in conventional deep learning training, then propose a novel idea – *Crowdlearning*, which decentralizes the heavy-load training procedure and deploys the training into a crowd of computation-restricted mobile devices who generate the training data. Finally, we propose *SliceNet*, which ensures mobile devices can afford the computation cost and simultaneously minimize the total communication cost. The combination of *Crowdlearning* and *SliceNet* ensures the sensitive data generated by mobile devices never leave the devices, and the training procedure will hardly disclose any inferable contents. We numerically simulate our prototype of *SliceNet* which *crowdlearns* an accurate DNN for image classification, and demonstrate the high performance, acceptable calculation and communication cost, satisfiable privacy protection, and preferable convergence rate, on the benchmark DNN structure and dataset.**

I. Introduction

Deep learning [1] outperforms other machine learning techniques in dramatically improving state-of-art in speech recognition [2], object detection [3] , face recognition [4] and genomics [5]. It specializes in capturing nonlinear features from complex data and stays resistant to the interference of irrelevant noise. The promising performance requires large quantity of training data and more complex neural network structure, which usually requires a server with powerful computation capability to train for days. Additionally, in order for a deep neural network (*DNN*) to perform well, it requires training dataset' distribution resembles that of the test dataset, and the best way so far to achieve such training dataset is to collect and analyze the user-generated data. Therefore, the current practice among service providers is to collect the user-generated data and provide relevant services for free in return.

However, conventional deep learning is performed in a centralized way where a server possesses and uses all the training data collected from the individuals. This raises serious privacy issues because user-generated data in its original form contains various sensitive information describing individuals [6] [7] [8]. Besides, the central server holds the ultimate decision on the training purpose in conventional deep learning, and therefore the server may privately train and learn more information than necessary. For example, image datasets provided for the face recognition may be additionally fed to a DNN which trains and learns location information based on the image background at the server's side. Such misbehavior is unobservable because the server holds ultimate control on what will be the input/label pairs used in training.

In the past decades, the number of mobile devices has exploded and device owners create massive data on daily basis. Such abundant user-generated data has given birth to the application of various predictive models based on machine learning techniques, *i.e.*, users' daily keyboard input is useful to build personal associational input predictive model; users' taken pictures are useful for face recognition and local gallery profiling. However these data are too sensitive to get exposed , in which case will heavily breach the data producer's privacy.

Inspired by the idea of *crowdsourcing* [9] and *crowdsensing* [10], we present a novel approach to bypass the data collection from mobile devices and directly train the DNN models with mobile devices' joint computation in a decentralized way. We propose *Crowdlearning* which decentralizes the heavy-load *training* procedure and completes it with the collaboration among computation-restricted mobile devices who generate and possess the training data. With *Crowdlearning*, data are kept locally and individually in each mobile device, and mobile devices cooperate with each other to share the intermediate results of training procedures. The service provider only needs to collect tiny pieces of knowledge from the mobile devices to build up its global model without having access to any individual's data.

Enabling *Crowdlearning* with mobile devices brings many extra **challenges**. **Firstly**, training a DNN usually involves complex optimization, and neural network with complex structure and massive training data usually results in training for hours or days even for powerful servers, and it is much harder to use mobile devices in the training. **Secondly**, the large size of the DNN (can be as large as several GBytes) makes it challenging for mobile devices to store the architecture and manage the training as well. **Thirdly**, data traffic is expensive especially for mobile devices, therefore our design is limited by the communication overhead as well. **Finally**, connections among mobile devices are unstable, and disconnection of mobile devices may halt the training process or even negatively impact the learning accuracy. *Crowdlearning* needs to be robust against device disconnection.

Those challenges are addressed by our novel training archi-

Xiang-Yang Li is the contact author. The work is partially supported by China National Funds for Distinguished Young Scientists with No. 61625205, Key Research Program of Frontier Sciences, CAS, No. QYZDY-SSW-JSC002, NSFC with No. 61520106007 and No. 61572347, NSF CNS 1526638, NSF CNS 1343355.

tecture –*SliceNet*– which realizes the mobile-crowded privacy-preserving DNN training. Our consequent **contributions** are:

1) We propose a novel idea —*Crowdlearning* and an implementable model —*SliceNet* which is feasibly deployed in the resource-constrained mobile devices to facilitate the computationally intensive DNN training.
2) *Crowdlearning* alleviates privacy concerns by keeping all individuals' data in their own devices, and *SliceNet* further addresses it by ensuring that the joint computation among devices will not disclose any inferable contents.
3) *SliceNet* enables data owners to hold ultimate control on the training purpose and can privately keep the customized training model for offline usage.
4) *SliceNet* tolerates bad network environment and achieves a relatively high performance neural network model.

II. Backgrounds and Preliminaries

A. Deep Neural Network

Different types of Deep Neural Network (DNN) exist, and this paper focuses on the DNN for the classification. A DNN has three types of layers. 1) *input layer* captures the input features of the original data, such as pixels of images or gene sequences, which will be delivered to the next layer – hidden layer. 2) *hidden layer* is composed of several *neurons*, and typically more than three hidden layers exist in each DNN. Neurons (which are nodes in hidden layers) receive inputs from the previous layer and deliver the *activation function*'s output to the next layer. The first hidden layer receives the input from the input layer, and the last hidden layer delivers the output to the last layer of the neural network – output layer. Edges only exist between two adjacent layers, and weights are associated with all edges. For any neuron in any layer (except the input layer) having the incoming edges with the weights $\{w_i : i = 1, 2, \cdots, k\}$ from the previous layer with k neurons , its output is $f\Big(\sum_{i=1}^{k} h_i w_i\Big)$ where $\{h_i : i = 1, 2, \cdots, k\}$ is the outputs of k neurons in the previous layer and f is the *activation function*. 3) *output layer* unites the abstract features of the last latent layer and outputs a probability vector in which each element stands for the probability of the training data belonging to the corresponding class. The architecture of Convolutional Neural Networks (CNNs) slightly differ from DNNs' in that CNN replaces the fully connected layers with convolutional and pooling layers and applies weight sharing to further reduce the neural network's parameter size. Without specification, abbreviation DNN will include CNN henceforth.

B. Training DNN

Training of DNN involves nonlinear optimization. For supervised learning, certain distance of the true expected value and the predicted value, often called as *loss function*, is the objective function to minimize, and the most common approach is to use the *gradient descent* to update the weights iteratively. In short, *gradient descent* starts with random weights and updates them based on the gradient of the loss function such that the loss function keeps decreasing during the iteration until converging to a local minimum.

More specifically, the training can be divided into two steps—*feed-forward* and *back-propagation*. Feed-forward calculates the outputs of all neurons layer by layer (from the input layer to the output layer), where the input is denoted as $\mathbf{x}$ and the final output of the output layer is denoted as $\tilde{\mathbf{y}}$. Then, the error E (*i.e.,* the difference between $\tilde{\mathbf{y}}$ and the vector of true labels $\mathbf{y}$) is calculated, which is back-propagated through the network and calculates the gradients of all weights layer by layer (from the output layer to the input layer). During the back-propagation every weight is updated based on the calculated gradients.

Mini-batch Gradient Descent [11] is used to calculate the gradients in this paper. The training dataset $\{\mathbf{X}, \mathbf{Y}\}$ with m samples is divided into $\lceil \frac{m}{b} \rceil$ mini-batches $\Omega = \{\omega_k : k = 1, 2, \cdots, \lceil \frac{m}{b} \rceil\}$, each containing b samples. Let ℓ be the *loss function*. Then, the b samples in a mini-batch ω_k will produce b loss values $\{\ell_{ki} : i = 1, 2 \cdots, b\}$, and every weight w in the neural network is updated as follows:

$$w := w - \alpha \cdot \frac{1}{b} \sum_{i=1}^{b} \frac{\partial \ell_i}{\partial w}$$

where α is the learning rate. One full iteration of feed-forward and back-propagation on all input data is denoted as an *epoch*.

We denote $\mathbf{z}^{(i-1)}$, $\mathbf{a}^{(i)}$, $\mathbf{W}^{(i)}$ as the i-th layer's input and output vector, and weight matrix between i-th and $(i+1)$-th layer, respectively. In *feed-forward* stage, the i-th layer receives the outputs of $(i-1)$-th layer as its inputs, and then forward its outputs to $(i+1)$-th layer, in which case we have $\mathbf{z}^{(i-1)} = \mathbf{W}^{(i-1)} \cdot \mathbf{a}^{(i-1)}$ and $\mathbf{a}^{(i)} = f(\mathbf{z}^{(i-1)})$, where $f(\cdot)$ is the activation function. In *back-propagation* stage, the i-th layer receives the error back-propagated from $(i+1)$-th layer, calculates the gradients to update weights, and propagates its own error to $(i-1)$-th layer. The gradient of $\mathbf{W}^{(i)}$ can be calculated as:

$$\nabla_{\mathbf{W}^{(i)}} \ell = \frac{\partial \ell}{\partial \mathbf{W}^{(i)}} = \frac{\partial \ell}{\partial \mathbf{z}^{(i)}} \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{W}^{(i)}} = \delta^{(i)} \mathbf{a}^{(i)}$$

where $\delta^{(i)}$ can be viewed as the error back-propagated from $(i+1)$-th layer and $\delta^{(i)} = \delta^{(i+1)} \cdot \mathbf{W}^{(i+1)} \cdot f'(\mathbf{z}^{(i)})$. Apparently, in both *feed-forward* and *back-propagation* stage, i-th layer only need information exchange with its two adjacent layers — $(i-1)$-th and $(i+1)$-th layer. This is the theoretical foundation for our work.

III. Crowdlearning via SliceNet

A. Crowdlearning

We use *Crowdlearning* to denote crowded deep learning where the dataset for training is directly crowdsourced from the mobile devices who generate it. *Crowdlearning* differs from distributed deep learning in several aspects. In distributed deep learning [12], several servers with wired connection have shared access to the collected dataset for training, and the training procedure is performed in a distributed manner among these servers. In *Crowdlearning*, the global dataset for DNN training is composed of each individual generated data and

resides in every mobile device, and will never leaves the mobile devices since generated. Training for the global DNN is performed individually with a coordinator's orchestration. Another difference lies in that distributed deep learning trains model in powerful servers, while *Crowdlearning* trains model in resource-restricted mobile devices. One of the most prominent advantage of *Crowdlearning* is that the individual data remains in individual devices, therefor privacy implications are alleviated, but not throughly addressed. We will carefully address the concerns completely with *SliceNet* later.

B. SliceNet

We take a step forward from the concept of *Crowdlearning* and propose a novel pipelined DNN training model – *SliceNet*. *SliceNet* decomposes DNN into several simpler and smaller components that are computationally manageable by resource-constrained mobile devices. Specifically, input/output layer stays integrated and assigns to mobile devices (*dominator*) who contribute local data as training samples, and the hidden layers are decomposed and distributed to other mobile devices (*followers*) who contribute to computation for load sharing. In case of other devices' unexpected disconnection, we reserve several backup devices (*dummy*) for displacement. The orchestration of the training is done by a *parameter server*. The parameter server maintains the global DNN by receiving updates from mobile devices to update global weights. Therefore, the parameter server does not undertake any computation tasks. However, all communication among mobile devices will be relayed by this server since there is not likely to be direct communication channels among mobile devices. That being said, the role of parameter server can be played by anyone who has enough bandwidth capacity. Since the parameter server cannot know the contents of the relayed communication, we assume public/private key pairs have been distributed among the mobile devices to encrypt/decrypt the relayed communication for the sake of privacy. Figure 1 gives a straightforward illustration about *SliceNet*'s architecture.

C. Assumptions and Adversary Model

When the training updates the weights of DNN based on a *dominator*'s training data, the *dominator* is the only party who has privacy concerns at that time, since, in this particular phase, others' data is not involved at all and they only undertake the computation task. Therefore, throughout the training via *SliceNet*, we could assume the *dominator* is honest; but this *dominator* could also be the adversary when he switches role from *dominator* to *follower* in other phases. So we have to say *dominator* is semi-honest. He has no intention to violate privacy when he contributes his data as training samples, but may try to invade others' privacy after role exchange using historical records as background knowledge. The rest parties: other devices and the parameter server, are assumed to be non-cooperative semi-honest adversaries. That is, we assume there is no collusion attack, and we also assume the devices and the server will follow the protocol specification but will try to infer other parties' private information throughout the protocol.

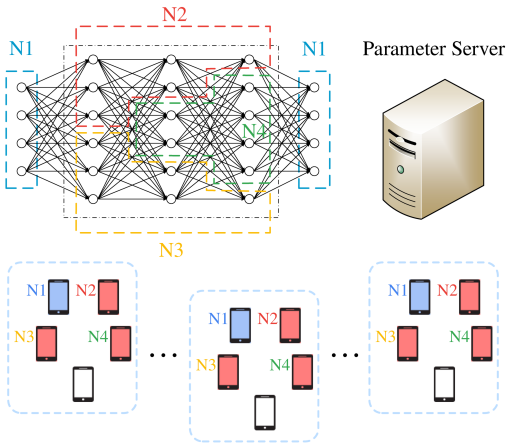


Fig. 1. **Example of *SliceNet***. Parameter server slices the network into four pieces (the upper graph) and assigns the pieces to mobile devices in each group (the lower graph). Blue devices (*dominator*), at this configuration, provide training data for training, thus possess the (input, output) layer, and red devices (*followers*) participates in the joint computation, thus possess the sliced hidden layer (the *id* N_i implies each mobile device's responsible slicing piece). Blank devices (*dummies*) are backup for blue/red devices' disconnection.

IV. Designing SliceNet

A. Overview

The incentive of *SliceNet* is to enable mobile devices to be affordable for complex DNN training, as well as to solve privacy concerns existed in conventional DNN training scenario. Considering one mobile device cannot tolerate the whole DNN training computation cost, can we reduce each device's load, and combine multiple devices together to train one model? So in the first stage, the parameter server slices the DNN into n pieces based on the estimation of overall computation cost and each mobile device's preferable computation capability (IV-B. *Neural Network Slicing*). We already give the analysis for the foundation that slicing DNN into pieces can still ensure the correctness and feasibility of training (II-B. *Training DNN*). However if there are many more devices than the number of sliced pieces, several devices will possess the same piece, which if not being appropriately dealt with could cause severe chaos. So in the second stage, the server logically divides the devices into several groups (IV-C. *Mobile Device Grouping*), where each group contains one *dominator* who indirectly provides personal data for training and several *followers* who perform hidden layer's computation for load sharing purpose. Subsequently, the training of DNN is performed independently within each group in a distributed manner, and the training knowledge (subset of the calculated weights) is also shared among the groups (IV-D. *Grouped Training*) to obtain a more generalized model. All groups' shared knowledge is relayed by the parameter server who aggregated and stores the collected knowledge to update the weights of the centralized DNN. (IV-E. *Global Updating*). After training, each mobile device can choose to privately keep their group's training model for usage without explicitly exposing inputs and outputs to the service provider.

B. Neural Network Slicing

For a neural network $\mathcal{N}$, we slice it into a set of pieces $\mathcal{S} = \{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_n\}$. Each $\mathcal{N}_i$ is an induced subgraph of $\mathcal{N}$, and $\mathcal{N} = \bigcup_{i=1}^{n} \mathcal{N}_i$. We assume n is known in advance and could be estimated according to DNN's overall load and device's acceptable load. Note that input and output layers will stay integrated and be assigned to one device for the correctness of the training, therefore we only consider the slicing among hidden layers ($\mathcal{N}$ denotes neural network removing input/output layers hereafter). Edges interconnecting the pieces are denoted as *cut edges* hereafter. Specifically, for cut edges, we let mobile users possessing the previous layer's neurons maintain and update the weights. We make such assignment because, for any previous/next layers, the output of the previous layer is the multiplication of a neuron's output and the weight, and letting the next layer's possessor know the weight will enable him to infer the output of the previous layer's neuron. Especially at the first layer this will directly disclose all the input data.

The incentive of slicing is to enable mobile devices to handle the computation cost of the pieces they are assigned. Intuitively, slicing the network into more smaller pieces will help decrease the computation cost for one mobile device, however too many small pieces will result in enormous communication cost, for that slicing the DNN will cut off many connected edges and isolate some adjacent layers, which will brings the communications between two mobile devices if their pieces are connected in the original DNN. With more pieces, there will be more edges cut off, and more communication overhead. On the other hand, the computation cost associated with each neuron depends on the activation functions as well as the edges incident to it, therefore not all neurons have the identical cost. We need to carefully design the slicing algorithm to balance the computation load assigned to every mobile devices, as well as to minimize the overall communication cost.

Cost Graph: We first build the cost graph according to the structure of DNN. The cost graph is used to balance the computation cost and minimize the overall communication cost. For a sliced piece, the main **computation** cost comes from two aspects: 1) Summation of each neuron's weighted inputs, in *feed-forward*. 2) Calculating the gradient of the loss function with respect to the connected edges' weights, in *back-propagation*. The main **communication** cost comes from one aspect: the transmission of the neurons' outputs (possessed by other devices) from the previous layer to the next layer through cut edges. Based on these observations, we can build the cost graph, where the vertex weight is assigned with each neuron's computation cost, and edge weight is assigned with communication cost. To formally model and solve the slicing problem, we first define the cost graph of a DNN $\mathcal{N}$.

Definition 1 (Cost graph): We define the cost graph $G_{cost} = \{N, E, W_N, W_E\}$ for a DNN $\mathcal{N}$ where N is the set of neurons, E is the set of edges, W_N is the set of computation costs associated with each neuron, and W_E is the set of communication costs associated with each edge.

$w_{n_{ij}} \in W_N$ is the computation cost required for neuron n_{ij} (j-th neuron in i-th hidden layer), which includes the computation of activation function, summation, and derivative calculation. $w_{e_{jk}^{h_i h_{i+1}}}$ is the communication cost for transmitting output from n_{ij} to $n_{(i+1)k}$ where $e_{jk}^{h_i h_{i+1}}$ denotes the edge connecting neuron n_{ij} and $n_{(i+1)k}$. Neuron n_{ij}'s weight $w_{n_{ij}}$ can be calculated by $w_{n_{ij}} = (\nabla_{\mapsto n_{ij}} + \nabla_{n_{ij}\mapsto}) \cdot \sigma_f$, where $\nabla_{\mapsto n_{ij}}$ is the number of edges incoming to neuron n_{ij}, $\nabla_{n_{ij}\mapsto}$ is the number of edges outgoing from neuron n_{ij} and σ_f is the approximation of activation function's cost. Edge e_{jk}'s weight $w_{e_{jk}}$ can be calculated by $w_{e_{jk}^i} = c$, where c is just a constant and can be simplified as 1 as edges' weights has no interactions with vertices' weights.

DNN Slicing As discussed, the objective of slicing is to minimize the overall communication cost (for the parameter server) with the constraint that each device's acceptable computation should be balanced (for the mobile devices). We formally define the DNN slicing problem:

Problem 1 (DNN slicing problem): Given a DNN $\mathcal{N}$ and the corresponding cost graph $G_{cost} = \{N, E, W_N, W_E\}$, the NN slicing problem is to partition the graph G into $S = N_1 \cup N_2 \cup ... \cup N_n$, with cutting edges E_{ij}^{cut} connecting N_i and N_j, such that:

$$\begin{aligned} \min \quad & \sum_{e \in E_{ij}^{cut}, 1 \le i < j \le n} w_e \\ \textbf{s.t.} \quad & \forall N_i, N_j \in S : \left| \sum_{\forall n \in N_i} w_n - \sum_{\forall n \in N_j} w_n \right| \le \eta, \end{aligned} \tag{1}$$

where η is the soft constraint over computation cost difference between any two mobile devices. Ideally, it could be 0.
This is a traditional weighted graph partition problem, which is widely used in parallel computation system design and sparse matrix multiplication.

We use an open tool METIS [13] to help implement the neural network slicing. In our setting, since we know transmitted weighted values will be summed as the next layer's inputs, we can actually reduce the communication cost by transmitting the sum instead of each value, from previous layer's possessor to next layer's possessor. This is an improvement in our problem settings to reduce the overall communication cost, and we adjust the costs in the cost graph accordingly. We can further leverage *quantization* and *vectorization* techniques to reduce the communication cost here.

C. Mobile Device Grouping

Assume there are many more devices (m) than the number of sliced pieces n ($m \gg n$), in which case many devices will be assigned the same piece. Because one iteration can only support one dominator's one batch of training samples, meaningless waiting or computation redundancy may occur. Computation and scheduling will become more complex as well. So we logically divide m devices into several groups, such that each group has one *dominator*, n *followers* and h *dummies*. The grouping strategy could be based on the geo-

graphical distance, social network, users' habits, *etc.*. *W.l.o.g.,* we assume grouping is random and the influence of the way devices are grouped is ignored in this paper.

D. Grouped Training

Obviously one group can train a standalone DNN model independently, but the distribution of one group's training data could differ greatly from the whole data's distribution. Other groups' training will not benefit from this group's data as well. So we propose two stages for grouped training: *intra-group training*, where devices in one group cooperate to train one DNN model with this group's training data, and *inter-group training*, where groups exchange very minor knowledge to improve each group's model generalization ability. One problem is that in one group only one *dominator* exists, who contributes his/her data as training samples. To make the most of all users' data, *shuffling withing the group* is further proposed to shuffle *dominator*, *followers* and *dummies* so that every device's data get a chance to be used as training samples.

Intra-group training: During the feed-forward, *dominator* calculates the weighted sums which are the inputs to the next layer, and these sums are delivered to the corresponding *followers* in the next layer. *followers* compute and deliver each weighted value to the next layer as well, which is iterated until the feed-forwarding reaches the output layer. All the delivery is relayed by the parameter server, for privacy concerns the contents are all encrypted therefore invisible to the server. We use asymmetric encryption that each mobile device generates a public/private key pair and then server distributes the public keys accordingly. Similar calculation and delivery are performed during the back-propagation as well. Notably, we pipeline the feed-forwarding and back-propagation to reduce the overall delay. The easiest way is setting up a timer to coordinate the parallel pipeline scheduling.

Shuffling within the group: After a pass (feed-forward and back-propagation) is finished for every input data possessed by *dominator*, devices in each group needs to be shuffled, so that another one could be *dominator* to contribute his data for training. Because the contents transmitted between devices are encrypted and invisible to the server, server can know each neural network piece's possessor without privacy compromise. So shuffling is implemented by the server, in the way that server first randomly chooses one mobile device as *dominator* without replacement, and then randomly assigns the slicing pieces to the rest devices as *followers*. Devices without assignment automatically become *dummy*. Each piece's weights should keep unchanged before and after shuffling and their delivery is relayed by the parameter server as well in the encrypted format after shuffling.

Therefore, there will be exactly $n-1$ shuffling after the initial grouping since there are n devices in each group and every device will take charge of the input/output layers exactly once. When all devices' input data has experienced exactly one pass after $n-1$ shuffling, an epoch is finished, and all the groups move to the next phase.

Algorithm 1: Group i's training in *SliceNet*

```
Each mobile user j downloads responsible piece N_j and the initial parameters
  w_j from parameter server;
for e = 1 to #epoch do
    Initialize all edges' cumulative gradient acc_{w_ΔG} to 0;
    for b=1 to #batch do
        Mobile devices within group i cooperate to calculate each edge l's
          gradient ΔG_l: ΔG_l = 1/batchsize Σ_k^{batchsize} ΔG_l^{(k)};
        forall edge e_l ∈ E do
            w_l = w_l − α * ΔG_l ;
            acc_{w_ΔG^l} += |ΔG_l|;
        end
    end
    Sort edges with acc_{w_ΔG^l} in descending order;
    Select the top θ proportion edges' weights to upload to the server;
    Download weights from server to replace group's local weights;
end
```

Inter-group knowledge sharing: We notice that SGD method always tries to find a direction towards the local minimum and then subtracts the product of learning rate and the calculated gradient from each weight. The closer to the local minimum, the smaller the absolute value of the gradient is. So we say for one edge e and its gradient ∂G_e, a large value of $|\partial G_e|$ indicates the edge e is more significant in finding the local minimum. Based on this observation and the idea of *dropout*, when a group finished an epoch, the devices select θ portion of edges' weights with largest gradients and upload the plaintext of them to the server. After then, the devices also request other groups' knowledge from the server (explained in next section).

We give an illustration of the logical pseudocode for group i in algorithm 1. Shuffling stage is omitted for logical integrity. The physical executions are implemented by devices in each group with aforementioned *mini-batch SGD*. Ending of each epoch, each mobile device checks the individually recorded responsible parameters' gradients and selects θ portion of responsible weights with largest gradient variance to upload to the server. Then each device retrieves some weights from the server to replace each local parameters.

Note that this is the only phase where the plaintext of trained weights are uploaded to the parameter server, which differentiates our work from [14] who releases the gradients to the server. We take the same strategy in [15] that weights are directly shared instead of the gradients. We believe directly releasing the gradients is much more dangerous, because solving the training data from equations parameterized by weights is much harder than solving equations parameterized by gradients. In addition, new weights are comprised of old weights, exchanged knowledge and accumulated gradients, so inferring training data from weights introduces more uncertainty than from gradients. Also uploading and downloading weights at the end of each epoch will further increase the difficulty for server to infer the relationship between gradients and input data. *SliceNet* does not require *differential privacy* considering the tradeoff between utility and privacy. We will show that *SliceNet* already achieves strong privacy protections even without *differential privacy* in the later section.

E. Global Updating

Parameter server maintains a global DNN model and updates its local parameters based on the received weights. For a specific edge in $\mathcal{N}$, multiple groups will submit multiple weights as their updates during the inter-group knowledge sharing, then the parameter server sets certain sliding window to calculate the average of the submitted updates within the sliding window, which is stored as the global parameter of the final DNN to be learned. To reduce the update frequency, we adopt the lazy updating method, which means the server will update its local parameters only when some mobile device sends the download request in inter-group knowledge sharing. When there is a download request for the weights, the parameter server will select the edges with uploading frequency (number of uploads) in sliding window's records larger than $\omega * \tau$ and send to mobile devices, where ω is the sliding window size and τ is the threshold to control the quantities of downloading. For any edge, its weight will be sent to the group if $\Gamma_{count} > \omega * \tau$ where Γ_{count} is this edge's uploading frequency within the sliding window.

To this point, it is clear that, although each group does not have direct access to others' datasets, but their training procedure captures the most significant knowledge by replacing their local weights with the server's global ones.

F. Privacy Analysis

We analyze what are visible to adversaries to analyze the privacy protection of our *SliceNet*.

Parameter server: In [14], gradients during the back-propagation are disclosed to the server which made it possible to calculate the values in the output layer as well as the input layer. Therefore the authors leveraged the exponential mechanism based on differential privacy. In our work, all intra-group training is invisible to the parameter server owing to the encryption, therefore only weights in inter-group knowledge sharing are visible to the server. Because a reported weight is the average of weights in each batch, it is not possible for the parameter server to infer the original weights. Recall that each weight is calculated from a pass of feed-forwarding and back-propagation of a single (input, label) pair, and that only θ proportion of edges are uploaded to the server, it is very challenging for the parameter server to infer the input from such limited knowledge. In addition, as discussed uploading weights instead of gradients greatly decrease the possibility to solve the training data from equations. By introducing more uncertainty, *SliceNet* can achieve high privacy protections without *differential privacy*, to get higher model utility.

As for the training purpose, it is the *dominator* who feeds the (input, label) to the intra-group training, therefore it is impossible for the parameter server to manipulate the training purpose directly. He has no ability to fake the users to train other DNN models against users' will. The only manipulation it may try is to send fake weights to the devices during the inter-group knowledge sharing phase, but we claim that it is not possible yet to intentionally manipulate the training goal only by manipulating several weights during the training. As DNN training procedure still operates like a *black box*, no one can manipulate several weights without seeing training data to get an accurate model for other purpose usage.

Since we bypass the data collection and all training data never leave the mobile devices since generated, users hold ultimate control over their data all the way without any worries about data exposure, storage intrusion, deletion verification, re-utilization, data trading, *etc.*.

Other mobile device: They have less knowledge about the weights in the global DNN than the parameter server, therefore it will be more challenging for any adversarial mobile device $\mathcal{A}$ to infer others' input based on this knowledge. However, each mobile device has access to the output delivered from the previous layer's possessor as well as the back-propagated error from the next layer's possessor. Considering that the number of neurons in the input layer is much greater than those in the first hidden layer, the possessor of the first hidden layer is not able to recover the input (more variables than the equations he has). Afterwards, what other next layers' possessors receive from the cut edges is the multiplication of the unknown output and the unknown weight, therefore a single adversarial device $\mathcal{A}$ is not able to recover other devices' neuron outputs in the feed-forward. There is shuffling and $\mathcal{A}$ may be assigned to the next layer by coincidence, at which point $\mathcal{A}$ knows old weights of the cut edges incident to the previous layer (belonging to the previous piece assigned to it). However, at this point, some weights have been updated already by the new device being in charge of the previous piece, and $\mathcal{A}$ does not know which have been updated, making its knowledge completely obsolete. Note that it is not even possible for $\mathcal{A}$ to calculate one single output confidently since it does not know which weight is not usable anymore. Considering that there is nothing available to confirm the calculated outputs based on his obsolete knowledge, $\mathcal{A}$ cannot successfully infer the outputs in this case neither. The same theory holds for the back-propagation as well, therefore $\mathcal{A}$ is not able to infer the training samples.

Additionally our work does not rely on any mix network like Tor [16] to hide the mobile devices' identities. *SliceNet* ensures mobile devices always have no idea whom they are communicating with, and the communication contents are always invisible to the parameter server. Remember we assume there is no collision attack between mobile devices and parameter server, thus this wouldn't introduce any other new privacy concerns.

V. Evaluation

A. Experiment Setup

We evaluate our system on two datasets frequently used for DNN benchmarking – MNIST and CIFAR-10 [17]. For MNIST, we use 60,000 images for training and 10,000 for testing, and we use 50,000 for training and 10,000 for testing for CIFAR-10. We adopted two kinds of deep learning model – Multi-Layer Perception (MLP), in which layers are fully connected, and CNN. MLP performs well with data having small feature space and CNN is specialized in capturing the complex data with large feature space. So we use MLP on

```
nn.Sequential {
  [input -> (1) ->  ... -> (13) -> output]
  (1): nn.Reshape(784)
  (2): nn.BatchNormalization
  (3): nn.Linear(784 -> 400)
  (4): nn.BatchNormalization
  (5): nn.ReLU
  (6): nn.Linear(400 -> 200)
  (7): nn.BatchNormalization
  (8): nn.ReLU
  (9): nn.Linear(200 -> 100)
  (10): nn.BatchNormalization
  (11): nn.ReLU
  (12): nn.Linear(100 -> 10)
  (13): nn.LogSoftMax
}
```

(a) MLP for MNIST

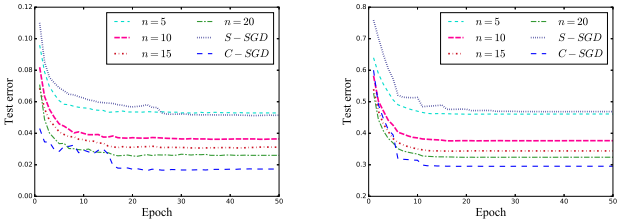
```
nn.Sequential {
  [input -> (1) -> ... -> (13) -> output]
  (1): nn.SpatialConvolutionMap
  (2): nn.ReLU
  (3): nn.SpatialMaxPooling(2x2, 2,2)
  (4): nn.SpatialConvolutionMap
  (5): nn.ReLU
  (6): nn.SpatialMaxPooling(2x2, 2,2)
  (7): nn.Reshape(3200)
  (8): nn.Linear(3200 -> 128)
  (9): nn.ReLU
  (10): nn.Linear(128 -> 64)
  (11): nn.ReLU
  (12): nn.Linear(64 -> 10)
  (13): nn.LogSoftMax
}
```

(b) CNN for CIFAR-10

Fig. 2. DNN architectures

TABLE I
TEST ERROR OF *SliceNet* FOR DIFFERENT SLICING NUMBER

n	S-SGD	5	10	15	20	C-SGD
MLP	5.15%	5.29%	3.63%	3.10%	2.62%	1.73%
CNN	46.88%	46.11%	37.65%	34.40%	32.40%	29.54%



(a) MLP for MNIST (b) CNN for CIFAR-10

Fig. 3. Convergence of *SliceNet* varies with slicing number n

MNIST dataset and CNN on CIFAR-10 dataset. Figure 2 shows the architecture of the two models.

Physically deploying *SliceNet* is temporarily unrealistic in our experiments since we do not have so many mobile devices, therefore we run 120 individual processes in a PC to simulate 120 mobile devices. The parameter exchange among devices is scheduled by *round robin* protocol. All the training samples are randomly uniformly distributed among all simulate processes.

For the comparison, we also implemented and tested two different settings: *centralized* model, which is the traditional training done by a central server, and *standalone* model, in which a single mobile user forms a group in *SliceNet*. In theory, the centralized model's DNN will be more generalized and reflect the characteristics of all dataset while the standalone's model is least general because the training model is a mixture of personal models highly biased to individual data.

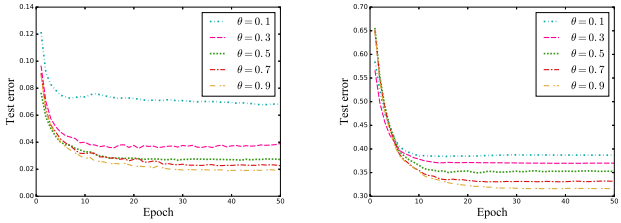
B. Slicing Number Evaluation

We varied the slicing number n among $\{5, 10, 15, 20\}$ for 120 devices, and NN slicing problem is solved by leveraging **METIS**. We randomly distributed the training samples into these 120 mobile users and group them into $\{24, 12, 8, 6\}$ groups respectively. θ and τ is set as 0.5 and ω is set as g to ensure the equal gain from per group. The test error is shown in table I. We also show the convergence rate for two models with different slicing number in figure 3. S-SGD stands for standalone-SGD and C-SGD stands for centralized-SGD.

We observe that with more slicing pieces, the performance of training model will get better, while the convergence rates (*i.e.,* at which epoch the error converges) are similar.

TABLE II
TEST ERROR OF *SliceNet* FOR DIFFERENT UPLOADING RATIO θ

θ	0.1	0.3	0.5	0.7	0.9
MLP	7.33%	3.83%	2.70%	2.37%	1.93%
CNN	51.41%	46.93%	35.35%	33.26%	31.57%



(a) MLP for MNIST (b) CNN for CIFAR-10

Fig. 4. Convergence of *SliceNet* varies with θ, when $\tau = 0.5, \omega = 10$

TABLE III
TEST ERROR OF *SliceNet* FOR DIFFERENT DOWNLOADING THRESHOLD τ

τ	0	0.2	0.4	0.6	0.8	1
MLP	3.07%	2.91%	2.70%	2.91%	3.14%	5.72%
CNN	35.89%	35.73%	35.35%	36.48%	38.07%	48.07%

This demonstrates that sacrificing certain communication cost does help improve the performance. The reason is that when mobile user number is fixed (overall training data is fixed), more slicing pieces require more users to constitute one group, which increases the intra-group training's occurrence and decreases inter-group knowledge sharing's occurrence. Thus each group's training model has higher generalization performance, which helps converge to the optimum model. Also, as we expected, S-SGD's performance is worst and C-SGD's performance is best.

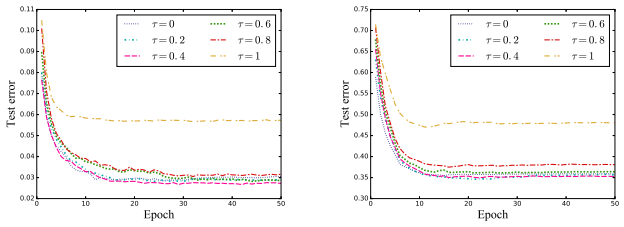
C. Generalization Performance Evaluation

To demonstrate the performance of our model, we compare our method with centralized model and standalone model. Uploading ratio θ controls the server's information gain from each group; downloading ratio τ controls each group's information gain from other groups; window size ω controls how many historical parameters are used to update global weights. We fix $m = 120$, $n = 12$, $g = 10$, and altered these three parameters.

In Table II, we varied θ and let $\tau = 0.5$, $\omega = 1 * g = 10$. As expected, larger uploading ratio indicates better generalization performance since sharing more knowledge between groups will increase global model's performance. The convergence is shown in figure 4. θ has no obvious influence over the convergence rate and *SliceNet* can converge quickly.

In Table III, we varied τ where 0 stands for downloading all weights existing in the sliding window, and 1 stands for downloading nothing from the server (no collaboration between groups). θ and ω are fixed as 0.5 and $1 * g = 10$, respectively. The relationship between τ and final performance is not clear except $\tau = 1$. For the rest values, the fluctuation in the test error is so small that it's not possible to conclude there is correlation between τ and the test error.

In Table IV, we varied ω and fixed $\theta = \tau = 0.5$. It appears that when $\omega \approx g$, *SliceNet* achieves the best performance. When ω is close to the group number, then server gains

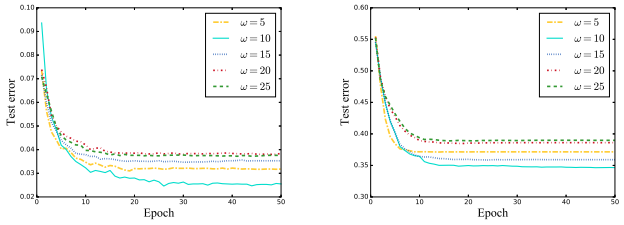


(a) MLP for MNIST (b) CNN for CIFAR-10

Fig. 5. Convergence of *SliceNet* varies with τ, when $\theta = 0.5, \omega = 10$

TABLE IV
TEST ERROR OF *SliceNet* FOR DIFFERENT WINDOW SIZE ω

ω	5	10	15	20	25
MLP	3.18%	2.53%	3.53%	3.82%	3.76%
CNN	37.15%	34.67%	35.91%	38.61%	38.97%



(a) MLP for MNIST (b) CNN for CIFAR-10

Fig. 6. Convergence of *SliceNet* varies with ω, when $\theta = 0.5, \tau = 0.5$

almost same knowledge from each group and updates global parameters without too much redundant historical records. Otherwise bias to some group may lead training model to highly co-adapt to the group's data, or redundant records will dilute the up-to-date weights' importance, which should have larger information entropy compared with old ones. We show the convergence in figure 6. When $\omega \approx g$, *SliceNet* achieves the lowest test error but converges relatively slower.

D. Computation and Storage Evaluation

We theoretically calculate the required computation and storage overhead and compare with the total load in traditional DNN. In our experiment, we use the same activation function – ReLU – for all hidden layers as its derivation is simple. We consider each neuron and weight has unit cost 1, and build the cost graph. Then we solve it with METIS. We list the theoretic cost model for one piece in Table V. #max shows the largest-cost among all the pieces, and #ideal is the average cost which is achieved when the slicing is perfectly balanced. The architecture of DNN is usually symmetric, thus the computation load is well balanced among the devices.

To demonstrate *SliceNet* can support mobile device's storage, we list the parameter size of original complete DNN and slicing pieces in Table VI. Slicing network greatly frees the storage load for each device and enables *Crowdlearning* among resource-restricted mobile devices.

E. Communication Evaluation

All the transmitted data format is *float* and size is 4B (Byte). For different slicing number n, we calculate the total communication cost for the parameter server and communication cost per device in one training sample in one iteration (Table VII).

TABLE V
COMPUTATION REQUIREMENT IN IDEAL SETTINGS AND IN *SliceNet*

(a) MLP for MNIST

n	5	10	15	20
#max	20948	10471	6977	5229
#ideal	20340	10170	6780	5085
ratio	1.03	1.03	1.03	1.03

(b) CNN for CIFAR-10

n	5	10	15	20
#max	362353	181176	120779	90587
#ideal	351782	175891	117260	87945
ratio	1.03	1.03	1.03	1.03

TABLE VI
STORAGE REQUIREMENT FOR COMPLETE DNN AND *SliceNet*

(a) MLP for MNIST

n		5	10	15	20
Complete	#node	1494	1494	1494	1494
	#edge	414600	414600	414600	414600
Max	#node	154	81	56	28
	#edge	22375	10338	6955	5172
dominator's	#node	794	794	794	794
	#edge	31360	31360	31360	31360

(b) CNN for CIFAR-10

n		5	10	15	20
Complete	#node	38154	38154	38154	38154
	#edge	2039936	2039936	2039936	2039936
Max	#node	8346	4756	3500	2697
	#edge	453792	259330	69533	123080
dominator's	#node	3082	3082	3082	3082
	#edge	313600	313600	313600	313600

TABLE VII
THEORETICAL COMMUNICATION COST(KB) PER TRAINING SAMPLE

(a) MLP for MNIST

n	5	10	15	20
#Comm	19.12	33.57	46.46	57.79
#Comm/device	3.18	3.05	2.90	2.75

(b) CNN for CIFAR-10

n	5	10	15	20
#Comm	214.87	331.45	818.11	1034.29
#Comm/device	35.81	30.13	51.13	49.25

One sample's size is 0.77KB and 3KB respectively in MNIST and CIFAR-10. From the result we see the overall communication is relatively large, compared with training sample's size, but each device shares an acceptable communication cost, almost at the same order with sample's size. In addition, slicing more pieces will increase the total communication load, but the cost for each device will decrease.

VI. RELATED WORK

Research of privacy-preserving learning has been an active work in machine learning community. Early works like secure multi-party computation (SMC) [18] can help protect training data, where data is split between multiple parties, who collaborate to train learning model over whole data, while no any single party can access to others' training data [19]. However SMC imposes too much computation overhead and is not applicable for deep learning. Differential privacy [20] is another popular approach, which directly perturbs original data, or perturbs gradients to update parameters [21], [22]. Some works also seek for tighter privacy loss budget

[23]. Differential privacy provides strict privacy protection, but introduces too much noise and degrades model's performance.

Homomorphic encryption is another kind of protections from encryption perspective. It enables training over encrypted data, without exposing the plaintexts [24]. Graepel *et al.* [25] focuses on finding training algorithms which can be implemented over encrypted data. Encrypting training data can protect users' privacy when using model's service while ensures returning approximately accurate outputs [26]. However homomorphic encryption only supports addition and multiplication operations, limits the supported operation times, and imposes heavy computation overhead, thus is hard to apply in deep learning.

Shokri *et al.* [14] provide a similar training scenario with ours, where training data is locally kept in each device who will train a local DNN model and share partial gradients with server to obtain the global DNN model. While it requires adding noise to gradients to protect privacy, which degrades performance. Our work doesn't rely on adding noise, and by sharing weights, instead of gradients, at the end of each epoch, instead of each iteration, our work further prevent inferring knowledge of training data from parameters' gradients.

To the best of our knowledge, we are the first to research how to directly deploy DNN training over resource-restricted mobile devices without privacy compromise. Some previous work [14], [15] claim their training scenario happens in mobile devices, but lacks the consideration of mobile devices' low computation and storage capacity and thus is impractical.

VII. Future Work and Conclusion

Our *SliceNet* strictly protects each participant's privacy during the whole *Crowdlearning*, but one limitation is that the parameter server is *single point of failure*, both from privacy and feasibility perspective. If the parameter server colludes with one *follower*, this will greatly threat *dominator*'s privacy. In addition, the global DNN model is maintained by the server and the communication also relies on the server. Our future work is to decentralize the functionality of the parameter server and achieve the resistant against the collusion attack. We will also develop the mobile APP for users to download to test and evaluate *SliceNet* in real mobile devices. We would like to investigate into more factors hasn't been included in this paper's experiment, like energy consumption, asynchrony.

In this paper, we successfully realized *Crowdlearning* with our novel architecture *SliceNet*. Notably, individual data never leaves the mobile device during *Crowdlearning*, and our rigorous analysis shows the joint computation in *Crowdlearning* does not leak individual data neither. Besides, individual data owners are given the ultimate control on the training purpose. Our extensive evaluation demonstrates *Crowdlearning* is feasible via *SliceNet* who achieves a high generalization performance and acceptable extra overhead for each device.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[5] M. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey, "Deep learning of the tissue-regulated splicing code," *Bioinformatics*, vol. 30, 2014.

[6] J. Qian, X.-Y. Li, C. Zhang, and L. Chen, "De-anonymizing social networks and inferring private attributes using knowledge graphs," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[7] J. Qian, X.-Y. Li, C. Zhang, L. Chen, T. Jung, and J. Han, "Social network de-anonymization and privacy inference with knowledge graph model," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[8] X.-Y. Li, C. Zhang, T. Jung, J. Qian, and L. Chen, "Graph-based privacy-preserving data publication," in *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*. IEEE, 2016, pp. 1–9.

[9] J. Qian, F. Qiu, F. Wu, N. Ruan, G. Chen, and S. Tang, "Privacy-preserving selective aggregation of online user behavior data," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 326–338, 2017.

[10] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges." *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, 2011.

[11] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 661–670.

[12] B. C. Ooi, K.-L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. Tung, Y. Wang *et al.*, "Singa: A distributed deep learning platform," in *MM*. ACM, 2015, pp. 685–688.

[13] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 1998, pp. 1–13.

[14] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1310–1321.

[15] H. B. McMahan, E. Moore, D. Ramage *et al.*, "Federated learning of deep networks using model averaging," *arXiv preprint arXiv:1602.05629*, 2016.

[16] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, 1981.

[17] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[18] O. Goldreich, "Secure multi-party computation," *Manuscript. Preliminary version*, pp. 86–97, 1998.

[19] Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining," *Journal of Privacy and Confidentiality*, 2009.

[20] C. Dwork, "Differential privacy," in *Automata, languages and programming*. Springer, 2006, pp. 1–12.

[21] M. Pathak, S. Rane, and B. Raj, "Multiparty differential privacy via aggregation of locally trained classifiers," in *Advances in Neural Information Processing Systems*, 2010, pp. 1876–1884.

[22] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *Journal of Machine Learning Research*, vol. 12, no. Mar, pp. 1069–1109, 2011.

[23] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," *arXiv preprint arXiv:1607.00133*, 2016.

[24] L. J. Aslett, P. M. Esperança, and C. C. Holmes, "Encrypted statistical machine learning: new privacy preserving methods," *arXiv preprint arXiv:1508.06845*, 2015.

[25] T. Graepel, K. Lauter, and M. Naehrig, "Ml confidential: Machine learning on encrypted data," in *International Conference on Information Security and Cryptology*. Springer, 2012, pp. 1–21.

[26] 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 *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 201–210.