Model Partition Defense against GAN Attacks on Collaborative Learning via Mobile Edge Computing

Cheng-Wei Ching, Tzu-Cheng Lin‖, Kung-Hao Chang‖, Chih-Chiung Yao, and Jian-Jhih Kuo*
Dept. of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan

***Abstract*—With growing concerns about privacy issues of machine learning, collaborative learning (CL) is developed to offer on-device training. However, adversarial behaviors of model inversion (MI) are undermining privacy of training data. Specifically, adversaries act as ordinary participants in CL and reproduce private data of a class in training data by training generative adversarial networks (GAN) on the fly, unknowingly. To this end, we design a novel model partition defense, PAMPAS, over user devices and trustworthy edge server to resist GAN attack, and formulate a new optimization problem, TENSOR, to optimize training time. To address the challenges that come with PAMPAS, we propose an algorithm TESLA that yields the optimal solution. Experiment and simulation results manifest that PAMPAS effectively defend GAN attack and TESLA reduces training time by 50% compared with other solutions.**

***Index Terms*—federated learning, GAN attack, model partition defense, mobile edge network**

I. Introduction

The rapid growth of collaborative learning (CL)[1] over user devices in mobile networks raises further concerns for model inversion (MI) [1]. The primary rationale behind CL is to train machine learning models on user devices with local data unexposed to central server [2]. However, sensitive data of victims (e.g., images) can be *unknowingly reproduced* via a *generative adversarial network* (GAN) by ordinary participants in disguise (i.e., adversaries) who take part in CL training phase (see Section II-A). Compared with MI in centralized learning, defending GAN attacks in CL is more challenging due to the reasons as follows: 1) *Updated parameters* of each iteration are exposed to adversaries who join the training phase. 2) Manipulated local model parameters are considered difficult to detect and eventually incur accuracy downgrade [3].

Many traditional MI attacks (not in CL) have been presented to steal sensitive information of training data. One is white-box attack where trained models are accessible to adversaries [4], [5]. Adversaries can gradually generate representative data similar to training data by establishing an attack model to mimic characteristics of training data. The other is black-box in which only prediction queries can be made by adversaries [6], [7]. Based on the different behaviors of models in the face of seen and unfamiliar data, adversaries are able to identify the differences and differentiate the members and non-members of training data by exploiting GANs. However, these attacks are found difficult to reproduce training data once models get as complex as are convolutional neural networks (CNNs) [6], [7]. Besides, *Differential Privacy* [8] (i.e., adding noises in data) has been proved effective to defend black-box attacks [9], [10]. In contrast, GAN attacks in CL are much more intractable since models and parameters updates are usually exposed to adversaries while noises added in data by Differential Privacy may completely collapse CL training phase [1].

To mitigate the issue, our idea is to leave the *entire* model *partially unknown* to users and employ *nearby* edge servers[2] to assist the partial training. That is, the model is partitioned into two parts, and users are responsible for training only one of them. Each layer of neural networks is able to be assigned to either user devices or edge server while each layer requires different *computation time* on different parties. Furthermore, two consecutive layers assigned to different parties should yield *transmission time*. In addition, to mitigate security concerns, the *input* and *output* layers should be trained on user devices to prevent the edge server from directly accessing personal data and labels. However, the state-of-the-art approaches of *model partition* emphasize the inference time minimization for trained models [11], [12]. None of them uses model partition to jointly 1) resist the GAN attack and 2) minimize the training time (including computation and transmission time) in CL scenarios, thereby providing the motivation of this paper.

Optimally partitioning the model into two parts to minimize total training time yet satisfying security concerns raises new research challenges as follows. 1) *Uncertain effect of layer concealing*. Model partition can prevent adversaries from knowing the entire model. However, concealing different layers from users (including adversaries) may lead to different performance of resisting GAN attacks (detailed in Section II). Thus, the set of layers assigned to user devices should be carefully determined to secure data privacy. 2) *Computation and transmission time trade-off*. Compared to user devices, powerful edge server can speed up the computation for acquiring new parameters of a local model. However, the size of data propagated between two consecutive layers should be addressed to avoid overlength transmission time, especially at layers with multiple branches. Moreover, more user devices may generate more intermediate data transmitted to the edge server for computation. 3) *Forward and backward propagation interplay*. During each epoch, the data are propagated forward to obtain the prediction loss and then backward to derive the new model parameters. The computation overhead and transmission overhead are different in the forward and backward propagation. Thus, the forward and backward propagation should be considered simultaneously.

To verify *uncertain effect of layer concealing*, we first implement a GAN attack method [1] to find the appropriate layers concealed from user devices. Based on the observation results, we present a novel **P**rivacy-**A**ware **M**odel **Par**tition Defense **S**ystem (PAMPAS) to defend against GAN attacks (in Section

* indicates the corresponding author; ‖ denotes the equal contributions.
Corresponding author's email: lajacky@cs.ccu.edu.tw

[1]CL includes distributed learning, federated learning, or decentralized learning, where several devices would take part in training phase [1].

[2]We assume that the edge server belongs to trustworthy third parties.

II). A new optimization problem **T**ime-**E**fficie**n**t **S**ecure M**o**del Pa**r**tition Problem (TENSOR) is formulated to optimize the system performance (in Section III). With the given parameters: 1) a CNN of CL, 2) computation overheads of each layer in forward and backward propagation, 3) transmission overhead from a layer to the next layers in forward and backward propagation, 4) computation capability of user devices and edge server, 5) transmission capacity between user devices and edge server, and 6) number of user devices, TENSOR asks for a set of *layers* concealed from user devices to minimize the training time in an epoch while ensuring that each layer is executed on appropriate party to secure data privacy. We then propose a **T**ime-**E**fficient **S**ecure **L**ayer Concealing **A**lgorithm (TESLA), to construct an auxiliary graph to find the optimal solution (in Section IV). Experiment and simulation results show that PAMPAS with TESLA can effectively defend against GAN attacks and minimize the training time with edge server compared with several naive methods (in Sections II-B and V).

II. Attack Model and Defense System

A. Attack Model

Fig. 1 shows the overview of GAN attack [1]. The adversary pretends as an ordinary user to join in CL for a given model[3] and receive parameters from the central sever[4] in each global epoch.[5] It secretly sets up a *GAN* where 1) the *discriminator* is a copy of the global model received from the central server and 2) the *generator* targets a class[6] in the global model and generates similar data that can fool the discriminator. Then, the generated data will be tagged with a *fake* label and merged with the local real dataset of the adversary to train the model. Eventually, a new but *biased* local update is uploaded to the central server for the aggregation with local updates from other users. Note that the local update of model parameters uploaded from the adversary is indistinguishable from that uploaded by legitimate participants. As a result, the global model becomes biased and intends to ask for more accurate information of the attacked class to enhance the feature extractors and fix the bias, whereas this helps privately-trained GAN of the adversary generate more human-readable images for the target class [1].

B. Effect of Concealing Different Layers

To observe the effect of concealing different layers from user devices, we implement a GAN attack method [1] with Python 3.7.6, TensorFlow 2.1.0, and Keras 2.3.1. We adopt MNIST dataset [14] of handwritten digits and AT&T dataset [15] of faces. MNIST dataset contains 70000 images of handwritten digits, and AT&T dataset includes 40 different faces, each of which has 10 images. Following [1], the number of output neurons in the CL models for MNIST and AT&T are set to 11 and 41, respectively. For the experiments on MNIST (or AT&T) dataset, we set both the learning rate for both the CL model and the generator to 1e-3 (or 2e-4), and their learning rate decay and batch size are set to 1e-7 (or 1e-8) and 64 (or 32). For each epoch, each ordinary user device takes 100 (or 10) images for each class to locally train the model. At the same time, the adversary targets a specific class and employs the CL model to be the discriminator to train its generator locally by generating 1000 (or 400) images of the target class for gradient update. The adversary then outputs 100 (or 10) images of the target class but tagged with a fake label (e.g., 11^{th} class for MNIST and 41^{th} class for AT&T) to locally train the CL model. All the users including the adversary upload the local updates to the central server for aggregation. The above operation is repeated until the generator of adversary can output human-readable images. All the models use Adam optimizer.

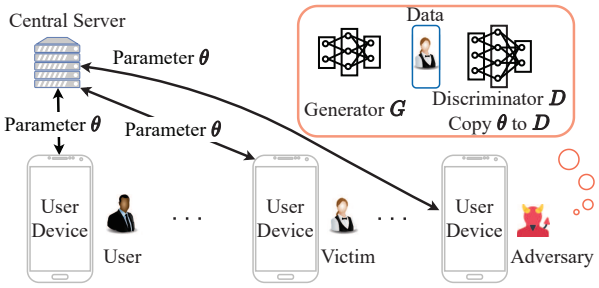


Fig. 1. Overview of GAN Attack on CL

Fig. 2 shows the images output by GAN attack in different epochs and the images appear clearer after 50 epochs. Then, different number of layers are concealed from user devices to examine the defense performance. Note that all user devices are not allowed to access the parameters of concealed layers. Intuitively, the more concealed layers tend to prevent the adversary from generating human-readable images. However, concealing the front[7] layers is much less effective than concealing the back layers as shown in Fig. 3. The generator can still capture some features to draw the face shadow when front layers are concealed from user devices, but the generated images become almost meaningless once a back layer is concealed. Thus, our system PAMPAS conceals *at least one back layer* from user devices and performs effectively as shown in Fig. 4.

C. Defense System - PAMPAS

PAMPAS is built over a mobile edge network which consists of user devices, edge servers, and a central server. The user devices (including the adversary) participant the training phase coordinated by the central server. The central server collects the information of user devices and edge servers (including computation capability and transmission capacity) before the training phase. Then, the central server selects a set of layers concealed from user devices. The first and the last layers of the model must be computed on user devices to prevent the edge servers from accessing personal images and labels. PAMPAS executes at least the *second-last layers*[8] on edge servers since at least 1 back layer should be concealed. Also, misplaced layers may prolong the overall training time. Therefore, our goal is to

[3]It is nature to assume that the model is a CNN, which is shown effective in image processing and object recognition.

[4]User devices can also use gossip algorithms to achieve the consensus model by averaging belief with neighbors locally without a central server [13].

[5]Here, global epoch means the exchange of newly updates of parameters, whereas local epoch means the training of each user device with its local data.

[6]The class can represent an infected organ in healthcare applications, a face of an individual in attendance system or a specific object in pattern recognition.

[7]Here the front (or back) layers are near the input (or output) layers.

[8]It may have 2 second-last layers since a model could have branching layers.

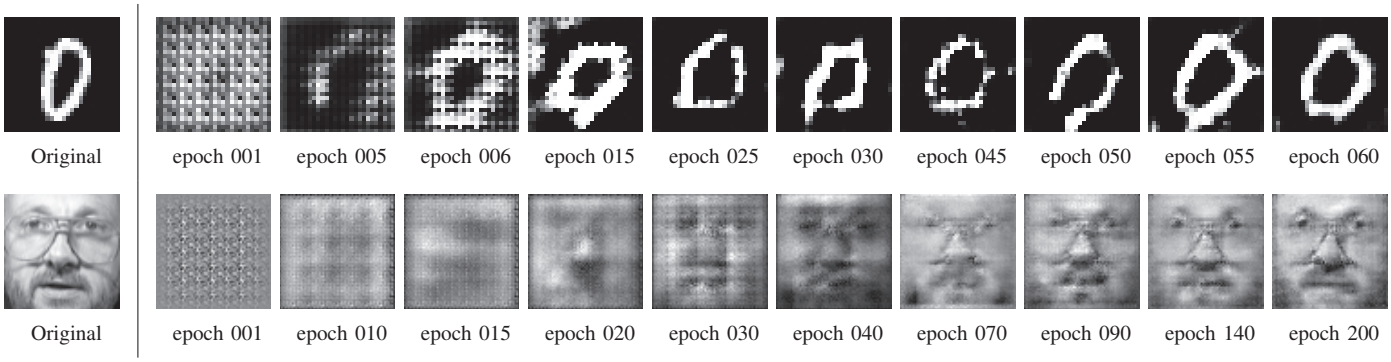


Fig. 2. Images generated in different epochs with GAN attack on CL for MNIST (Zero) and AT&T (face #13).

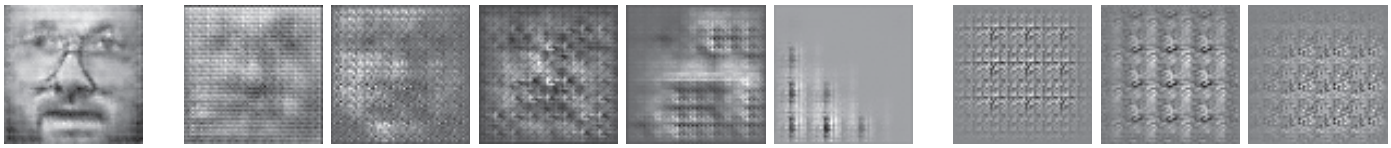


Fig. 3. Concealing different number of layers from the *front* or *back* to prevent adversaries from getting the entire model.

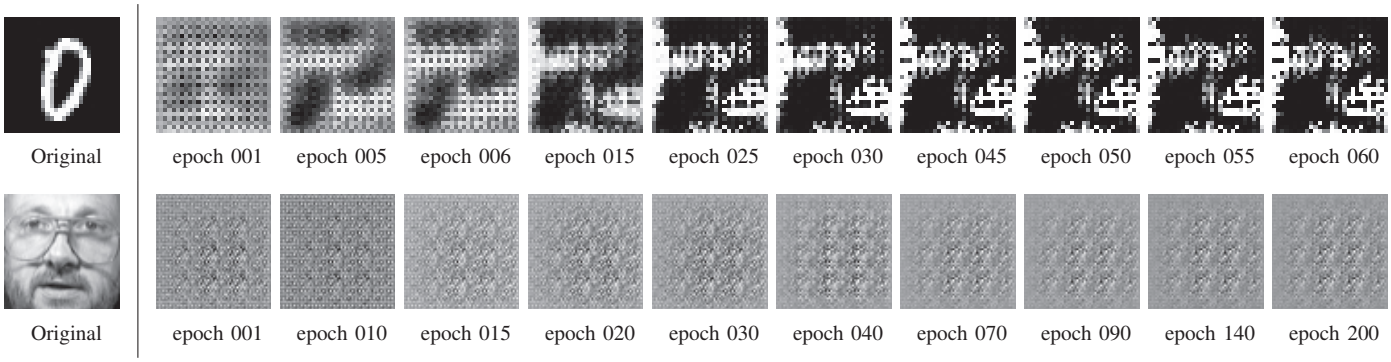


Fig. 4. Images generated in different epochs with GAN attack on CL in PAMPAS for MNIST (Zero) and AT&T (face #13).

balance the overheads between user devices and edge servers to minimize the training time while ensuring the data privacy.

III. The TENSOR Problem

This paper considers a mobile edge network that consists of: 1) n user devices (i.e., participants in CL) and 2) an edge server, where p^u and p^s denote the computation capability of an user device and the edge server, respectively, and r is the transmission capacity between a user device and the edge server.[9] The given CL model is a DAG $G = \{V, E\}$, where

1) V denotes layers in the CNN and each layer $v \in V$ has computation overheads $c^f(v)$ and $c^b(v)$ in the forward and backward propagation during an epoch, and
2) E denotes links between two consecutive layers in the *forward* propagation and each forward link $\overrightarrow{vw} \in E$ has transmission overheads[10] $d^f(v)$ and $d^b(v)$ in the forward and backward propagation during an epoch, respectively.

Thus, the computation time of layer $v \in V$ on user devices and edge server are in an epoch denoted by $t^u(v) = \frac{c^f(v)+c^b(v)}{p^u}$ and $t^s(v) = \frac{n(c^f(v)+c^b(v))}{p^s}$ since the edge server has to process the data from n user devices. Note that each layer could have two or more branching next layers. Then, the transmission time between layer v and its next layers in an epoch is denoted by $t^t(v) = \frac{d^f(v)+|\delta^+(v)|\cdot d^b(v)}{r}$, where $\delta^+(v)$ denotes the sets of out-degree links of node $v \in V$ in G. Following the settings above, the TENSOR can be formulated as an *integer linear programming* (ILP) as follows.

Let binary *decision variables* y_v^u and y_v^s respectively denote whether layer $v \in V$ is assigned to user devices or to the edge server. Each layer is executed on exactly one party, yielding

$$y_v^u + y_v^s = 1, \qquad \forall v \in V. \tag{1}$$

Let i and o denote the input and output layer in V. To avoid exposing the personal data and labels to the edge server, the layers i and o should be executed on user devices, i.e.,

$$y_i^u = y_o^u = 1. \tag{2}$$

Recall that concealing the second-last layers from user devices is more likely to resist the GAN attack (see Section II). Let $L \subseteq V$ denote the set of second-last layers. Thus,

$$y_v^s = 1, \qquad \forall v \in L. \tag{3}$$

[9]To explore the intrinsic properties of the TENSOR, we assume that the computing capability and transmission capacity of user devices are identical.

[10]For out-degree links from layer v, the data in *both* forward and backward propagation are proportional to the number of neurons in layer v [16].

TABLE I
MODEL INFORMATION

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
$c^f(v)$	60	70	10	40	10	40	10
$c^b(v)$	50	45	15	25	5	30	30
$d^f(v)$	8	30	25	10	30	5	0
$d^b(v)$	1	0.6	0.8	0.4	1	0.2	0

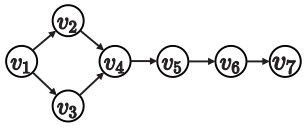


Fig. 5. A given model DAG.

Then, all the layers will be partitioned into two sets, and the transmission of layer v is required if its next layers are assigned to another party different from that of v. Let binary *decision variable* x_v denote whether to transmit the data of layer v via wireless communication. The relations are shown as follows,

$$x_v \geq y_v^u - y_w^u, \qquad \forall v, w \in V : \overrightarrow{vw} \in E \tag{4a}$$
$$x_v \geq y_w^u - y_v^u, \qquad \forall v, w \in V : \overrightarrow{vw} \in E \tag{4b}$$
$$x_v \leq y_v^u + y_w^u, \qquad \forall v, w \in V : \overrightarrow{vw} \in E \tag{4c}$$
$$x_v \leq 2 - y_v^u - y_w^u, \qquad \forall v, w \in V : \overrightarrow{vw} \in E \tag{4d}$$

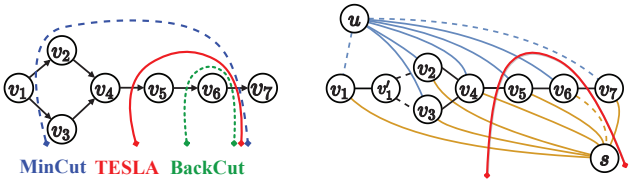
Therefore, the problem can be formally defined as follows.

Definition 1. Given a model DAG $G = (V, E)$ associated with computation overhead $c^f(v)$ and $c^b(v)$ and transmission overhead $d^f(v)$ and $d^b(v)$ for each $v \in V$ in an epoch, computation capabilities p^u and p^s of each user device and the edge server, and transmission capacity r between user device and edge server, the **T**ime-**E**ffici**e**n**t** **S**ecure M**o**del Par**t**ition Problem (TENSOR) finds a set of layers concealed from user devices, satisfy the constraints (1)−(4), and minimize the training time including computation and transmission time, i.e.,

$$\text{minimize} \quad \sum_{v \in V} x_v \cdot t^t(v) + \sum_{v \in V} (y_v^u \cdot t^u(v) + y_v^s \cdot t^s(v)) \tag{5}$$

Example 1. The example shows the effect of different model partition strategies for TENSOR, where V has seven nodes (i.e., layers from v_1 to v_7 in Fig. 5). Table I summarizes the given values of computation and transmission overhead of each node.[11] Assume that the computation capabilities of each user device p^u and the edge server p^s are 25 and 100 GFLOPS, the transmission capacity r is 2 Mbps, and 20 user devices take part in the CL. Fig. 6(a) compares TESLA with two intuitive approaches: 1) concealing only second-last layers from user devices (BackCut) and 2) minimum cut (MinCut). BackCut only assigns the second-last layer v_6 to the edge server and the computation time is $(60+70+10+40+10+10)/25 + (50+45+15+250+5+30)/25 = 14.8$ on user devices and $20 \times (40+30)/100 = 14$ on the edge server and transmission time $(30+1)/2 + (5+0.2)/2 = 18.1$, yielding training time $14.8 + 14 + 18.1 = 46.9$. By contrast, MinCut selects a set of links with the total minimum transmission time 7.6 and only assigns layer v_1 to user devices. Then, the computation time is 6 on user devices and 58 on the edge server. Thus, the elapsed time for an epoch is 71.6. Finally, TESLA (detailed in Section IV) assigns layers v_1, v_2, v_3, v_4, and v_7 to user devices, which only requires computation time 14.2 on user devices and 17 on edge server and transmission time 7.8. The training time for an epoch is 39. Compared with BackCut and MinCut, TESLA reduces training time by 16% and 46% in an epoch. ■

[11]Note that last layer v_7 has no next layer and thus $d^f(v_7) = d^b(v_7) = 0$.



(a) Results of different strategies. (b) Auxiliary graph $\mathcal{G}$ of TESLA

Fig. 6. Example of TENSOR. (a) Three lines represent different model partition strategies. (b) The dashed links have infinite weights.

IV. ALGORITHM DESIGN

Recall that the traditional CL executes all the layers on user devices (UnCut). However, UnCut does not conceal any layer from user devices and thus the adversary can reconstruct the images of a specific target class with GAN attack. Two naive approaches can be applied to TENSOR on PAMPAS. One is concealing only second-last layers from user devices (BackCut). The other one is to directly employ the minimum cut (MinCut) [17] to partition the model (except the last layer) into two parts by cutting a set of links with the minimum total transmission time. However, both of them neglect the *computation and transmission time trade-off* and *forward and backward propagation interplay*, and the overlengh computation or transmission time may prolong overall training time.

To solve TENSOR, we propose an algorithm named TESLA to carefully address both the challenges simultaneously. The idea is to construct an *undirected auxiliary graph* $\mathcal{G}$ to decide a set of layers concealed from user devices. The graph $\mathcal{G}$ first clones G to have the layers and links of the model. Besides, two extra *computation nodes* are added into $\mathcal{G}$ to represent user devices and edge server, respectively. To determine the suitable party for each layer, TESLA introduces two *computation links* into $\mathcal{G}$ to connect each layer's node to the two computation nodes, respectively. Thus, cutting one computation link for a layer indicates that the layer is *not* executed on the corresponding party. Accordingly, the weight of computing link connecting to the computation node of user devices (or edge server) is set to the computing time of executing the layer on the edge server (or user devices). Similarly, for each link from node v to node w in G, the weight of the *transmission link* between nodes v and w in $\mathcal{G}$ is set to their data transmission time. Note that some layers may have more than one next layers in G. To deal with the cases, TESLA removes the transmission links between v and the nodes of next layers in $\mathcal{G}$, and then creates a *virtual node* v' to connect v to the nodes of its next layers via *virtual links* instead. The weight of the link between v and v' is set to the total transmission time between layer v and its next layers, whereas the weights of links between v' and the nodes of next layers are set to *infinite*. TESLA then finds a set of links with the minimum total computing and transmission time to partition the nodes of $\mathcal{G}$ into two components.

A. Algorithm Description

TESLA includes the following three phases: 1) Virtual Node (VN) Augmentation, 2) Computation Link (CL) Connection, and 3) Minimum Cut (MC) Construction. VN Augmentation makes a clone of the model DAG in the auxiliary graph and

adds *computation nodes*, *virtual nodes*, and weighted *virtual links*. CL Connection adds weighted *computation links* in the auxiliary graph. Finally, MC Construction finds a set of links with the minimum weight to partition the nodes in the auxiliary graph into two components and admits the optimal solution.

1) Virtual Node (VN) Augmentation: Based on the model DAG $G = \{V, E\}$, VN Augmentation starts to construct the *undirected* auxiliary graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$. For each node $v \in V$, VN Augmentation creates a clone node v into the auxiliary $\mathcal{G}$. Then, for each node v in G with *only one out-degree* link $\overrightarrow{vw} \in \delta^+(v)$, it connects the two clone nodes v, w in $\mathcal{G}$ with an undirected link with a weight of the transmission time $t^t(v)$. For each node v in G with more than one out-degree links (i.e., $|\delta^+(v)| \geq 2$), it creates a *virtual node* v' into $\mathcal{G}$. Then, it

a) connects v' and v via an undirected link with a weight of the total transmission time $t^t(v)$, and
b) connects v' and each out-degree neighboring node w in $\mathcal{G}$ via an undirected link with an *infinite* weight.

Lastly, VN Augmentation creates two *computation nodes* u, s into $\mathcal{G}$ to represent users devices and edge server, respectively.

Example 2. Fig. 6(b) follows Example 1 to demonstrate VN Augmentation. It adds an undirected link in $\mathcal{E}$ for each $v \in V$ with $\delta^+(v_2) = 1$, e.g., $\overline{v_2 v_4}$. Then, it creates v_1' for v_1 and then adds links $\overline{v_1' v_1}$, $\overline{v_1' v_2}$, and $\overline{v_1' v_3}$ with weights $t^t(v_1)$, ∞, and ∞, since $\delta^+(v_1) = 2$. Finally, nodes u and s are created. ■

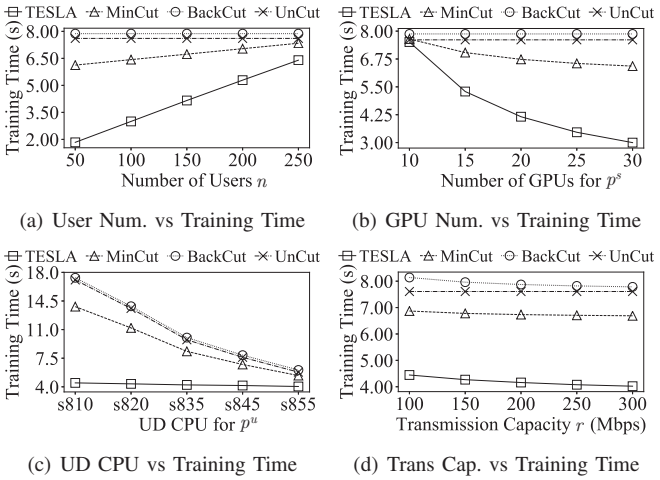
2) Computation Link (CL) Connection: For each node v in the model DAG G, CL Connection connects its corresponding node v in $\mathcal{G}$ to the *computation nodes* u, s via an *undirected* link e in $\mathcal{E}$ with a weight $c(e)$ of the computation time of the corresponding layer executed on *the other party*. That is,

$$c(e) = \begin{cases} t^s(v), & \text{if } e = \overline{vu} \text{ and } v \notin \{i, o\}; \\ t^u(v), & \text{else if } e = \overline{vs} \text{ and } v \notin L; \\ \infty, & \text{otherwise.} \end{cases} \tag{6}$$

It can be envisaged that the computation link from node v to node u (or s) indicates that the layer is executed on the user devices (or edge server). Recall that the first and the last layers are executed on user devices for data privacy, whereas the second-last layer(s) (i.e., L) are executed on the edge server. Therefore, CL Connection sets the weight of the computation link between the layer's node and its *desired* computation node to *infinite* to avoid assigning the layer to the inappropriate party.

Example 3. Fig. 6(b) follows Example 2 to show CL Connection. It connects u and s with $\mathcal{V}$. Links between u and each node in $\mathcal{G}$ are generated, namely, $\overline{uv_1}, \overline{uv_2}, \overline{uv_3}, \overline{uv_4}, \overline{uv_5}, \overline{uv_6}$, and $\overline{uv_7}$, and have weights ∞, 23, 5, 13, 3, 14, and ∞. The links between s and each node in $\mathcal{V}$ follows up while their weights are set to 4.4, 4.6, 1, 2.6, 0.6, ∞, and 1.6. ■

3) MC Construction: MC Construction employs the existing minimum cut algorithm (e.g., Stoer–Wagner algorithm [17]) to find a set of links to partition the nodes in $\mathcal{G}$ into two components by setting nodes u and s as the source and sink. The layers executed on user devices and edge server will be in the component of u and s, respectively. Also, the *cut* transmission links indicates the data transmission will occur between user devices and edge server for these layers.



(a) User Num. vs Training Time (b) GPU Num. vs Training Time

(c) UD CPU vs Training Time (d) Trans Cap. vs Training Time

Fig. 7. Effect of different parameters on different metrics. (ResNet152-V2)

Example 4. Fig. 6(b) follows Example 3 to find the minimum cut (i.e., the red curve in Fig. 6(b)) by MC Construction. The cut links are $\overline{v_1 s}$, $\overline{v_2 s}$, $\overline{v_3 s}$, $\overline{v_4 s}$, $\overline{v_4 v_5}$, $\overline{uv_5}$, $\overline{uv_6}$, $\overline{v_6 v_7}$, and $\overline{v_7 s}$. That is, only layers v_5 and v_6 are executed on the edge server. The required computation and transmission time are $4.4 + 4.6 + 1 + 2.6 + 1.6 + 3 + 14 = 31.2$ and $5.2 + 2.6 = 7.8$, yielding 39 in total. ■

B. Theoretical Analysis

Time Complexity. The overall time complexity is $O(|E| + |V| \log |V|)$. The details are omitted due to page limit. ■

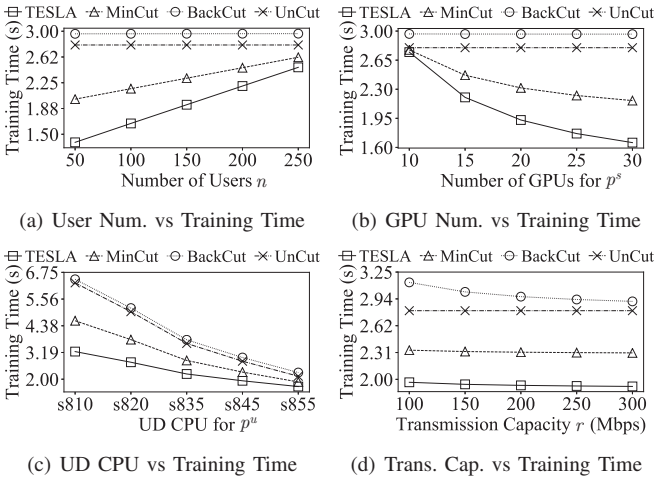
Theorem 1. TESLA yields the optimal solution of TENSOR.

Proof. We first prove that the auxiliary graph $\mathcal{G}$ constructed by VN Augmentation and CL Connection ensures that the corresponding node v of each layer is connected to either computation node u or s by link $\overline{vu}$ or $\overline{vs}$ in $\mathcal{G}$ after the set of cut links removed in MC Construction. That is, each layer is assigned to either user devices or edge server. We prove it by contradiction. Assume that there is a node $v \in \mathcal{G}$ such that two links $\overline{vu}$ and $\overline{vs}$ are either in the set of cut links or not in that set. Clearly, the latter case leads to a contradiction immediately since u and s are still connected via v in this case. It suffices to consider that both $\overline{vu}$ and $\overline{vs}$ are in the cut. Without loss of generality, assume that u and v are not connected in $\mathcal{G}$ after the cut links are removed. In this case, link $\overline{vs}$ can be removed from the set of cut links and there is still no path connecting s and u. That leads to another contradiction and the above statement holds. Then, the minimum cut constructed in MC Construction counts all link weights of any two neighboring nodes in $\mathcal{G}$ that are assigned to different components. The total weight of the selected of links in MC Construction is equal to the sum of computation and transmission time (i.e., training time) and it is guaranteed to be the minimum total weight. Therefore, the theorem follows. □

V. Performance Evaluation

A. Simulation Setting

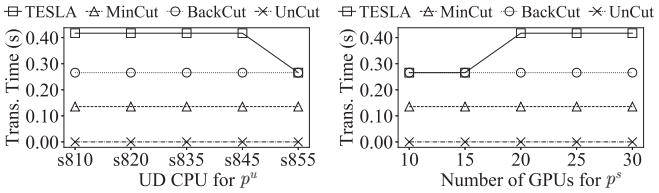
TESLA is compared with baseline UnCut and traditional approaches BackCut and MinCut in a network where n user



(a) User Num. vs Training Time (b) GPU Num. vs Training Time

(c) UD CPU vs Training Time (d) Trans. Cap. vs Training Time

Fig. 8. Effect of different parameters on different metrics. (MobileNet-V2)



(a) UD CPU vs Trans. Time (b) GPU Num. vs Trans. Time

Fig. 9. Effect of computation capability on transmission time. (ResNet152-V2)

devices and an edge server join in the CL for two models, ResNet152-V2 [18] and MobileNet-V2 [19]. The computation capability of a user device p^u is set based on the benchmarks of Snapdragon CPUs from s810 to s855, whereas that of the edge server p^s is set based on the number of GTX1080Ti GPUs available. The transmission capacity between a user device and the edge server r ranges from 100 to 300 Mbps. The number of user devices n ranges from 50 to 250. The above parameters are changed to observe the metric of training time in an epoch.

B. Comp. and Trans. Time & Forward and Backward Prop.

Overall, the training time in an epoch decreases as p^u, p^s, and r increase while increasing as n is getting more as shown in Figs. 7 and 8. BackCut performs similar to UnCut and has the longest training time since it only assigns the second-last layer(s) to the edge server so that user devices handle most computation. In contrast, MinCut aims to minimize the transmission time (see Fig. 9) so that it benefits from the powerful edge server and has shorter transmission time. Note that UnCut does not employ the edge server and thus has no transmission time but is under the threat of GAN attack. Instead, TESLA takes *computation and transmission time trade-off* and *forward and backward propagation interplay* into account at the same time and carefully examines the relation among p^u, p^s, r, and n to obtain the optimal solution.

C. Relation between Trans. Time and Comp. Capability

Fig. 9 also manifests the non-trivial results that the transmission time decreases and increases as the computation capabilities of user devices and edge server strengthen, respectively. These phenomena have a lot to do with the architecture of CNN models that usually have more data propagated in front layers since the front convolutional layers extract the features from raw data (e.g., images). Then, as the computation capability of a user device grows, TESLA tends to assign more layers to user devices, which lowers the data size for transmission. Similarly, as the computation capability of edge server grows, more layers are assigned to the edge server to raise the transmission time.

VI. Conclusions

In this paper, we examine the effect of model partitioning on GAN attacks. To tackle uncertain effect of layer concealing, we present a novel model partition defense system PAMPAS. To jointly balance computation and transmission time trade-off and address forward and backward propagation interplay, an optimization problem, TENSOR is investigated in depth. We propose an efficient algorithm TESLA to subtly augment graph and assign link weights so as to find an appropriate cut on a given model to optimize training time, including computation and transmission time of layers. Extensive experiment and simulation results manifest that PAMPAS is effective to defend GAN attacks and TESLA can further reduce 50% of training time compared with the variants of traditional algorithms.

References

[1] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the GAN: information leakage from collaborative deep learning," in *ACM CCS*, 2017.

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017.

[3] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *USENIX Security*, 2020.

[4] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton, "A methodology for formalizing model-inversion attacks," in *IEEE CSF*, 2016.

[5] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *ACM CCS*, 2015.

[6] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *IEEE SP*, 2017.

[7] U. Aïvodji, S. Gambs, and T. Ther, "Gamin: An adversarial approach to black-box model inversion," *AAAI Workshop on PPAI*, 2020.

[8] C. Dwork, "Differential privacy," in *ICALP*, 2006.

[9] L. Zhao, Q. Wang, Q. Zou, Y. Zhang, and Y. Chen, "Privacy-preserving collaborative deep learning with unreliable participants," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1486–1500, 2019.

[10] L. Jiang, X. Lou, R. Tan, and J. Zhao, "Differentially private collaborative learning for the iot edge," in *EWSN*, 2019.

[11] J.-I. Chang, J.-J. Kuo, C.-H. Lin, W.-T. Chen, and J.-P. Sheu, "Ultra-low-latency distributed deep neural network over hierarchical mobile networks," in *IEEE GLOBECOM*, 2019.

[12] C.-C. Hsu, C.-K. Yang, J.-J. Kuo, W.-T. Chen, and J.-P. Sheu, "Cooperative convolutional neural network deployment over mobile networks," in *IEEE ICC*, 2020.

[13] K. Zhang, Y. Liu, J. Liu, M. Liu, and T. Başar, "Distributed learning of average belief over networks using sequential observations," *Automatica*, vol. 115, p. 108857, 2020.

[14] Y. LeCun and C. Cortes. "MNIST handwritten digit database". [Online]. Available: http://yann.lecun.com/exdb/mnist/

[15] F. S. Samaria and A. C. Harter, "Parameterisation of a stochastic model for human face identification," in *IEEE WACV*, 1994.

[16] S. Lee, A. Agrawal, P. Balaprakash, A. Choudhary, and W.-K. Liao, "Communication-efficient parallelization strategy for deep convolutional neural network training," in *ACM/IEEE SC Workshop on MLHPC*, 2018.

[17] M. Stoer and F. Wagner, "A simple min-cut algorithm," *J. ACM*, vol. 44, p. 585–591, 1997.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.

[19] M. Sandler *et al.*, "MobileNetV2: Inverted residuals and linear bottlenecks," in *IEEE CVPR*, 2018.