An Energy-efficient and Privacy-aware Decomposition Framework for Edge-assisted Federated Learning

YIMIN SHI and HAIHAN DUAN, School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen and Shenzhen Institute of Artificial Intelligence and Robotics for Society, China
LEI YANG, School of Software Engineering, South China University of Technology, China
WEI CAI, School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen and Shenzhen Institute of Artificial Intelligence and Robotics for Society, China

Deep Learning (DL) is an essential technology for modern intelligent sensor network and interactive multimedia applications, having problems with user data privacy when training on a central cloud. While Federated Learning (FL) motivates to preserve user privacy, it also causes new problems of lower user terminal usability and training efficiency, which caused substantial energy consumption. This article proposes a novel energy-efficient and privacy-aware decomposition framework to improve user-side FL efficiency under pre-defined privacy requirements with the assistance of Mobile Edge Computing (MEC) and Software Decomposition. It takes the propagation of each neural layer as the migrating unit and considers the tradeoff relationship between privacy and efficiency. We also propose an online scheduling algorithm to optimize the framework's training performance. Furthermore, we summarize eight privacy-sensitive information classes on which existing privacy attacks base and design configurable privacy preservation mechanisms for each class. Simulations and experiments prove the effectiveness of our framework and algorithm in FL efficiency improvement and the effects of different privacy constraints on the overall training efficiency.

CCS Concepts: • **Theory of computation** → **Distributed algorithms**; • **Security and privacy** → *Privacy protections*; • **Computing methodologies** → Machine learning;

Additional Key Words and Phrases: Mobile edge computing, software decomposition, federated learning, distributed computing

ACM Reference format:
Yimin Shi, Haihan Duan, Lei Yang, and Wei Cai. 2022. An Energy-efficient and Privacy-aware Decomposition Framework for Edge-assisted Federated Learning. *ACM Trans. Sensor Netw.* 18, 4, Article 53 (November 2022), 24 pages.
https://doi.org/10.1145/3522741

This work was supported by Project 61902333, 61972161 supported from National Natural Science Foundation of China, and Project AC01202101038 from Shenzhen Institute of Artificial Intelligence and Robotics for Society.
Authors' addresses: Y. Shi, H. Duan, and W. Cai (corresponding author), School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, 2001 Longxiang Boulevard, Longgang District, Shenzhen, Guangdong, China and Shenzhen Institute of Artificial Intelligence and Robotics for Society, Xinghe World, 1 Yabao Rd, Longgang District, Shenzhen, Guangdong, China; emails: {yiminshi, haihanduan}@link.cuhk.edu.cn, caiwei@cuhk.edu.cn; L. Yang, School of Software Engineering, South China University of Technology, Guangzhou Higher Education Mega Center, Panyu District, Guangzhou, Guangdong, China; email: sely@scut.edu.cn.

1550-4859/2022/11-ART53 $15.00
https://doi.org/10.1145/3522741

1 INTRODUCTION

At present, sensor network and interactive multimedia applications in the artificial intelligence era are becoming more comprehensive, including a series of application scenarios such as intelligent sensor network, mobile **augmented reality (AR)**, mobile intelligent video editing, smart live streaming auditing, and mobile games. Typically, **Deep Learning (DL)** [10] technologies are commonly used in recent years, including but not limited to ubiquitous sensing, **Computer Vision (CV), Natural Language Processing (NLP)**, intelligent video/audio processing, and Big Knowledge-based [12] models. These applications often require cloud computing as additional computing power to support the computation of deep neural network. However, the cloud-centric DL training process will put additional pressure on the backbone network and cannot guarantee the privacy of training data provided by users. Currently, the emergence of **Mobile Edge Computing (MEC)** and **Federated Learning (FL)** have, respectively, solved part of the problems. In MEC, edge devices are fixed devices close to the user's mobile terminal devices and relatively rich in computing power. Furthermore, edge devices can be equipped with specialized parallel processing units with a lower energy-efficiency ratio like GPUs. The edge device can push DL services closer to the mobile terminal to avoid backbone network communication, reducing user delay and backbone network pressure to improve the efficiency and reduce the energy consumption.

While FL is a successful instance of **Software Decomposition (SD)** and migration, it decomposes the original training program executes on the central cloud into two parts and migrates the gradient generation part to the user device where it locates the privacy-sensitive training data. The cloud executes the model aggregation part. In this way, terminals only need to transfer the generated gradients to the cloud, avoiding transferring the training data on the backbone network, protecting user privacy. However, one significant property of FL is offloading the cloud's training workload to terminals and increasing their computation pressure. Since the computing power of terminals is generally constrained and uneven, while DL models are becoming more complex and heavy, the traditional application of FL will face the following two significant challenges: First, on the side of users, their regular usage of terminal devices will be affected. Since DL model training is resource-consuming, FL training will occupy terminals and leave the regular user requests in resource starvation, and the heavy computation will cause unaffordable energy consumption for terminals. Second, the overall training efficiency will also be problematic on the cloud side, because the eventual time consumption of an FL training round largely depends on the bottleneck participants with minor computation capabilities. This shows that FL essentially is trading the training efficiency and device availability for user privacy.

Edge devices in MEC have lower latency than the cloud and a relatively better energy-efficiency ratio than terminals. Therefore, MEC can be again introduced as a solution to improve the terminal FL training efficiency to reduce the entire energy consumption. However, utilizing external devices that can be untrustworthy means that we are re-finding the balance between privacy and efficiency and hope to trade the least privacy costs for the most efficiency improvements. In this way, the deployment of procedure components becomes a new problem. Simply offloading the entire procedure of the terminal model's **Forward Propagation (FP)** and **Backward Propagation (BP)** to edges will cause serious privacy threats and performance defects. From the privacy perspective, this requires the terminal to expose excessive privacy-critical information to the edge and network, even including primary training samples. Direct offloading can neither guarantee training efficiency, since the edge resource starvation can also easily happen when all terminals offload the whole procedure to the terminal.

Existing practices to solve a similar privacy preservation problem while using edge devices include hiding the feature extractor from the rest of the model [18] or applying the differential privacy in the inter-device communication [14]. Nevertheless, these methods have limited scalability and can not guarantee the effectiveness for complex distributed DL systems with finer-grained training components. Current works [17] optimizing the FL efficiency focus more on the selection of terminal participants, aiming to reduce the affect of inefficient terminals on the overall training performance. However, the potential assistance of MEC has not been fully discussed. Other work [26] about the edge-assisted efficient FL needs to change the original model structure according to the setting of edges, which loses the generality and compatibility and does not consider privacy issues. In addition, the above works do not discuss the potential tradeoff relationship between privacy and efficiency. Therefore, they fail to achieve higher efficiency, which is based on the prior knowledge of user requirements on their privacy and their trusting condition on edges. To solve the efficiency and usability problems mentioned above and fully utilize the tradeoff relationship between privacy and efficiency, we need to practice the software decomposition to decouple the original FL model and achieve sufficiently elastic migration strategies. We also need to design a component scheduler to optimize the efficiency according to user privacy requirements.

This article proposes a novel energy-efficient FL decomposition framework that works in general multi-edge and multi-terminal scenarios by implementing the layer-based model decomposition and privacy-aware online scheduling algorithm, which optimizes the training efficiency according to the configurable user privacy requirements to decrease the energy consumption. The layer-based model decomposition causes less communication costs between components and consistency requirements. Additionally, the layer-based decomposition can provide the scheduler with more finer-grained components and more elastic migration strategies. The scheduler in our framework executes the privacy-aware online scheduling algorithm, which eliminates the gap between the predictive estimation and the actual condition and balances the privacy and efficiency. To ensure that FL training can still effectively preserve user privacy to the required extent while assuming the edge device untrustworthy, we summarize the pivot information classes on which the current representative privacy leakage attacks are based. Moreover, we design proper privacy-preserving mechanisms for these information classes during the scheduling and make the privacy exposure more a performance tradeoff than a pure risk.

The contributions of this article are as follows:

(1) To the best of our knowledge, this is the first work proposing a privacy-aware decomposition framework for the edge-assisted FL in a general multi-terminal and multi-edge scenario, effectively considering the tradeoff relationship between the privacy and the efficiency.
(2) We have analyzed and classified the privacy-sensitive information utilized by current representative privacy attacks and proposed specific highly configurable preservation strategies for each of them.
(3) A privacy-aware online scheduling algorithm is also proposed to solve the efficiency optimization problem for the framework under fixed privacy condition, which can eliminate the negative influence of system's unpredictable randomness.
(4) The effectiveness of the framework is verified through simulations. Impacts of different configurable privacy hyperparameters are visualized, the privacy-efficiency tradeoffs are analyzed and discussed.

We organize the remaining sections of this article as follows: In Section 2, we do a brief literature review on efficient FL, Distributed DL, and privacy attacks in both FL and general DL. The framework design and problem formulation will be studied in Section 3, while Section 4 discusses the privacy-preserving online scheduling algorithm. Section 5 will evaluate the framework and algorithms through a set of simulations, and Section 6 concludes the article.

2 LITERATURE REVIEW

2.1 Efficient Federated Learning

Many recent works start to emphasize that the computation and power limitation of certain user devices may negatively affect the overall FL model training efficiency. Based on this, a variety of specific methods has been proposed to optimize the FL performance. These works can be basically classified into two representative categories. One standard methodology is to improve the training efficiency by utilizing tradeoffs between it and some other features in the traditional FL framework. The proposed optimization problem FEDL [21] focuses on the tradeoff between the learning time and the user equipment's power consumption, while the FedCS [17] trades the model robustness and overall accuracy for the training efficiency through the client selection algorithm in each round. What is more, in the FL paradigm named PruneFL [9], it uses the model parameter pruning to raise the training efficiency, which is also trading the model accuracy with efficiency. FedGKT [6] proposes to re-distribute the neural network on the data owner and server, requiring to change the model structure. Another sample [15] proposed a new parameter compression method and a distributed updating algorithm, which is also more or less sacrificing the eventual model accuracy. Some other adaptive approaches [23] mentioned in a current survey [11] achieve this by changing the frequency of the global aggregation. Another representative strategy to improve the training efficiency is to achieve the help from extra computing resources like edge computing devices. One recently proposed framework, EdgeFed, introduces the edge device into the traditional training framework, which keeps the computation of the first few layers on the terminal devices but offloads the computation of the remaining layers to the edge. On the edge, all terminal layers' outputs will be aggregated into a larger matrix as the input of the remaining layers. However, the elasticity of its FL model decomposition is constrained, since it requires users to re-implement the model structure and training procedure, and hasn't discussed how to deal with the new privacy risks brought by those external untrustworthy edge devices. Another edge-assisted FL framework [20] based on layer-decomposition lacks the mathematical modeling and effective privacy-aware scheduling algorithm for more general MEC scenarios with multiple terminals and multiple edges. These two works do not discuss the new tradeoff between user privacy and efficiency brought by edge devices and therefore fail to utilize user privacy requirements as prior knowledge to improve training efficiency further.

2.2 Distributed Deep Learning

At present, the application of distributed computing in the field of traditional DL and data mining has two major motivations. One is to improve the model training efficiency through some parallelism approaches. The other one is to preserve the user privacy of the data owner during the period of the transfer learning or inference. The previous works which focus on the efficiency-motivated distributed deep learning can be further classified into the following three classes: data parallelism, synchronous, and asynchronous pipeline parallelism. Some decentralized methods have been previously proposed to mine the correlation among sensor events in a parallel way [25]. GPipe [8] and DAPPLE [1] are representative works of synchronous pipeline-based parallelism of DL while asynchronous approaches include PipeDream [5]. One of the properties shared by pipeline-based approaches is that both the computing workloads and communication delays should be first evenly decomposed into similarly sized components with fixed numbers and generally assume that the processing environment is relatively constant. For the motivation of privacy preservation, especially in the application scenario where the major training process happens on the external devices instead of the data generator's, some recent implementations [14, 18, 22] vertically divide the original DL model into two parts and deploy them on different devices. The first part, named feature extractor, generally includes some convolutional layers in the head of the entire model, which is kept on the data generator's device, while the second part includes all the following parts and is deployed on the cloud or edge. Some technologies like differential privacy may be applied during the intermediate results' transfer. However, the similar privacy protection implements may be less valid for some newly proposed attacks and is less elastic in the DL model decomposition.

2.3 Privacy Threats in FL and DL

The representative privacy attacks in the field of FL and traditional DL can be briefly summarized in the the following categories: Model-access-based Inversion, GAN-based Inversion, Gradients-based Inversion, Recovery from Representation, and Recovery from Incomplete Inputs. The Model-access-based attacks [3, 4] generally utilize the identity of the victim and partial features of his training data to inverse more privacy-sensitive information about the victim's data through accessing a trained DL model. Some work [3] managed to partially reconstruct the appearance of the victim with only his name (identity) by accessing a white-box facial recognition model. While in some representative works of GAN-based Inversion attacks [7, 24], the attacker can train his own GAN model during the model training process to deceive the victims to expose more privacy information about their datasets. Specifically, the mGAN-AI [24] model can effectively attack the data owners in the FL scenario. The representative works in Gradients-based inversion attacks [27, 30] are able to utilize the gradients generated by the FL participants during the training phase to reconstruct the original input through optimization processes. For example, the recent famous attack DLG [30] can reconstruct the image input of a training participant in a pixel-wise accuracy in an ambiguous round of FL. Some work [13] in the Recovery from Representation attack can partially recover the original input of the model by utilizing the intermediate results on the deeper layers and the structure information of the model, which could raise new threats for the distributed training systems with layer-based or vertical decomposition. Some other works [29] can recover the privacy from incomplete inputs, which raises risks, especially applying a horizontal decomposition of the DL models in some distributed DL training systems [28].

3 FRAMEWORK OVERVIEW

This section explains the layer-based decomposition of the DL model and the proposed framework architecture, including terminals, edges, Blackboard, and the scheduler.

3.1 Layer-based Model Decomposition

There are two candidate methods for the software decomposition on the FL models: vertical decomposition [14, 18, 20, 22] and horizontal decomposition [29]. For the vertical decomposition, the model is exclusively decomposed into at least two layer sets, while all layers in the same set are sequentially connected. For the horizontal decomposition, if the model is divided into n different parts, then the number of layers in each part will be the same as the original model, but those layers only exclusively account for $1/n$ of the original layer. In our proposed framework, we apply the layer-based vertical decomposition, taking each layer as a single set, for the following reasons: (1) The dependency relationship in the vertical decomposition will be less complex, making optimization modeling and solving easier. Generally, in any propagation phase, any layer L will only communicate with its previous layer $L-1$ and next layer $L+1$. (2) There is no need for extra consistency control among different components in the vertical decomposition. Components that belong to the same user only need to be executed in a sequential order one after another. (3) The layer-based decomposition treats every single layer as the minimal inseparable unit to be offloaded and scheduled, which provides the scheduler with a large number of finer-grained components and thus a wider optimization space.

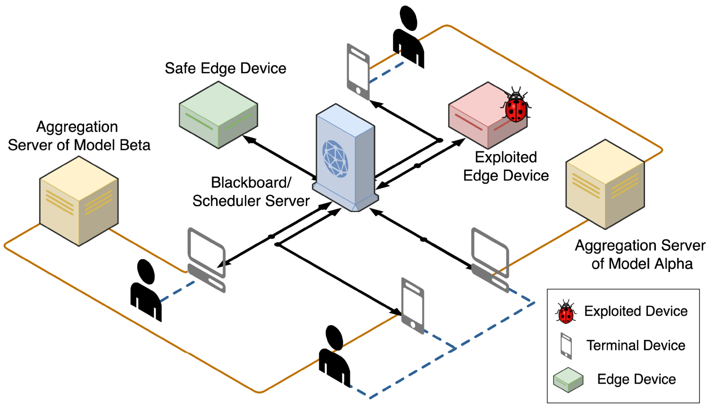
Safe Edge Device
Aggregation Server of Model Beta
Blackboard/ Scheduler Server
Exploited Edge Device
Aggregation Server of Model Alpha
Exploited Device
Terminal Device
Edge Device

Fig. 1. Framework architecture with multiple terminals and multiple edges; each terminal may train different models. The information confidentiality of tasks deployed on exploited devices will be damaged by attackers.

3.2 Multi-terminal Multi-edge FL Framework

In the proposed framework shown in Figure 1, we see there are multiple terminal devices and multiple edge devices. From the hardware perspective, we assume a difference exists between terminals and edges: It is more practical to equip edges with specialized parallel processing units like GPUs, which can provide higher floating-point operations resources and a significantly lower energy-efficiency ratio than terminal devices. Since the framework offloads the computation workload from the terminal's CPUs to the edge's GPUs, it decreases the energy needed by the same amount of **floating-point operations (FLOPs)**. What is more, we allow different user terminals to train different user models. Therefore, there will be multiple implicit aggregation cloud servers. In the proposed framework, we have only changed the training procedure at the user side and left the communication behavior between the parameter aggregator and the terminal unchanged.

All edge devices should provide standard configurable micro-services for all common DL layers' FP and BP computation procedures and intermediate results caching. The standard DL layer includes the dense layer, convolutional layer, pooling layer, activation functions, and so on. All the communication between two devices will pass through an anonymous transponder server named "Blackboard", where the global scheduler is located on. The global scheduler aims to optimize the overall training efficiency. In each training round, each layer-based component can be assigned by the global scheduler to a candidate device that can maximize its execution efficiency, or equally, minimize the average waiting time of the completion of corresponding computing tasks. The candidate device set includes the owner terminal of the model and all edge devices that can pass specific privacy constraints.

For example, if a DL layer L is allocated on edge E in its FP procedure, then edge E will receive the output from the corresponding device D that was assigned the previous layer $L-1$ in the same round, and then compute the FP output of layer L according to the hyperparameter and weights w_L given by L's owner's device. After that, E will cache all the intermediate results of L in its local memory or disk and pass the output data of layer L to the next device F on which the next layer $L+1$ is allocated. Then, in the BP of L in the same round, the gradient of the output of L will be passed back to E by F, and E can then compute the gradients of w_L and the gradient of its input, then further pass it back to its previous device D.

Table 1. Notations for the Framework Modeling and Problem Formulation

T	Set of all terminal devices.	Δt	Hyperparameter, the time limitation in the weight and gradient exposure measurement.
E	Set of all edge devices.	$C_{m,l}$	Computation resource cost of a specific layer in FLOPs.
L_m	Layers in a specific model.	$T^c_{m,l}$	Computation time cost of the layer on a device.
$\tau^{lyr}_{m,l}$	Information tuple describes a specific model layer.	$F_{m,l}$	Communication time cost of the layer in the forward propagation phase.
τ^{dev}_i	Information tuple describes a computing device.	$B_{m,l}$	Communication time cost of the layer in the backward propagation phase.
τ^{task}_k	Information tuple describes a specific task.	T^B_k	Time interval between a task's arrival and publishment.
$\bar{f}, \sigma^2_f$	Parameters define the distribution of the operation frequency of the device.	$T^Q_{k,i}$	Queuing time of a task on a specific device.
$\bar{p}, \sigma^2_p$	Parameters define the distribution of the network speed of the device.	K	Set of all tasks.
Ω_m	Hyperparameter, the user-defined threshold on the model structure exposure measurement.	$Q_{k,t^{pub}}$	Set of tasks will be executed before a specific task on the same device.
Γ_m	Hyperparameter, the user-defined threshold on the weight and gradient exposure measurement.	$D^t_{u,l}$	Set of devices where a task can be assigned to at a specific time.

4 MODELING AND PROBLEM FORMULATION

In this section, we will first mathematically model the various environment entities and randomness in the proposed framework, then design a series of privacy preservation constraints for it by summarizing the existing most representative privacy attacks, and finally present a privacy-aware scheduling optimization problem according to the general multi-terminal and multi-edge application scenario of the framework, expecting to maximize the training efficiency according to different users' privacy requirements. Important notations are collected in Table 1 for convenience.

4.1 Environment Modeling

We assume that the users and terminal devices that initially participate will not disconnect with the scheduler during the training procedure. Therefore, we represent the terminal devices with a constant set T. Similarly, we represent the edge devices with a constant set E. Intuitively, we have $T \cap E = \emptyset$. To present the structure of different DL models, we use a set L_m to present the layers of model m as in Equation (1).

$$L_m = \{1, 2, 3, \ldots, l_m\} \tag{1}$$

The l_m is the total layer number of the user model m. For each DL layer in each user model, we have a layer configuration tuple to represent its properties as in Equation (2).

$$\tau^{lyr}_{m,l} = (s_{in}, s_{out}, s_w, type, hps)_{m,l} \tag{2}$$

The s_{in} is the input size of the layer, the s_{out} is the output size, s_w means the size of the layer's weight. The *type* in the layer configuration tuple means the DL type, including the Dense Layer, Convolutional Layer, and others. *hps* means the other hyperparameters related to the layer type, for instance, the convolutional layer has the hyperparameters such as kernel size and padding size. The configuration tuple of layers will be used in the scheduling procedure to measure the computation and communication delays.

To represent the computing power of device, similar to the layer configuration tuples, we define the device configuration layer as in Equation (3).

$$\tau^{dev}_i = (\bar{f}, \sigma_f, \bar{p}, \sigma_p)_i \tag{3}$$

Since in the actual condition the specific processing environment of a computer fluctuates over time and is difficult to predict accurately, we can represent the computation power of a device as a random value with certain probability distributions instead of a constant. In the proposed framework, we assume the floating-point operation frequency of device i belongs to a Gaussian distribution where $\bar{f}$ is the mean of the random variable and σ_f stands for the standard deviation of the floating-point operation frequency. In Equation (4), the t in f_t means the floating-point operation frequency of device at a specific time t.

$$f_t \sim \mathcal{N}\left(\bar{f},\, \sigma_f^2\right) \tag{4}$$

The properties of the network can also be defined similarly. Since all devices do not directly communicate with each other in our framework and all the message packages pass through the central Blackboard, we summarize the network bandwidth between the device and the Blackboard into the configuration tuple of the device. We assume that the network bandwidth between device i and Blackboard at any time slot t belongs to a Gaussian distribution where the $\bar{p}$ is the mean of the random variable and the σ_p stands for its standard deviation as shown in Equation (5).

$$p_t \sim \mathcal{N}\left(\bar{p},\, \sigma_p^2\right) \tag{5}$$

The set of devices where the layer l of terminal u's model m can be assigned at time slot t is represented by $D_{u,l}^t$. In the case where privacy constraints are not considered, the $D_{u,l}$ includes the terminal device u, and all the edge devices $d \in E$. When considering the privacy constraints, the $D_{u,l}$ will generally include the owner device u and a subset of edges.

Before the formal formulation of the optimization problem, we first need to model the tasks. The tasks here mean the basic atomic unit of forward or backward propagation procedure of layers that can be offloaded between terminals and edges. In other words, every task is based on a specific layer of the original DL model, with a particular phase indicator that identifies the task in either FP or BP process. Similar to the device and model, we use a task tuple to model the tasks and express its information essential to our optimization problem as in Equation (6).

$$\tau_k^{task} = \left(t^a, \tau_{m,l}^{lyr}, i^{last}, i^{asg}, \pi, u..\right)_k \tag{6}$$

The t^a means the arrival time of the task k, the $\tau_{m,l}^{lyr}$ represents the corresponding layer l in model m the task bases on. i^{last} generally means the previous device index that is assigned with layer $l-1$ in model m in the same round, while i^{asg} means the index of currently assigned device to execute this specific task k. π indicates the propagation phase, which can be either FP or BP. u means the owner terminal index of this task. We define K as the set including all the tasks.

4.2 Configurable Privacy Constraint

In the proposed FL learning framework, the original user models are decomposed into multiple layer-based components, some of which would be offloaded to the edge devices during each local training round as tasks. This framework introduces edges as a new category of participating devices. The trust situation of edges is different. Some public edges are more likely to be threatened by malware and attackers and become untrustworthy to users, while those private edges that can be only accessed through LAN could be more trustworthy. Moreover, users' privacy importance of the training set also differs from each other. Those untrustworthy devices with high-importance datasets raise new privacy challenges and need specific privacy preservation strategies to be designed in the framework. Furthermore, we hope these mechanisms to be configurable and elastic enough to meet different user privacy requirements. The primary logic chain of developing the privacy protection mechanisms in our framework is shown in Figure 2 and the third part of our literature review in Section 2. We first summarize the existing five categories of privacy attack methods for FL and general DL, then defined eight major privacy-sensitive information classes that are often utilized directly by the attack methods to steal user privacy. Finally, we propose a privacy protection mechanism for each of these information classes. Some of those mechanisms are

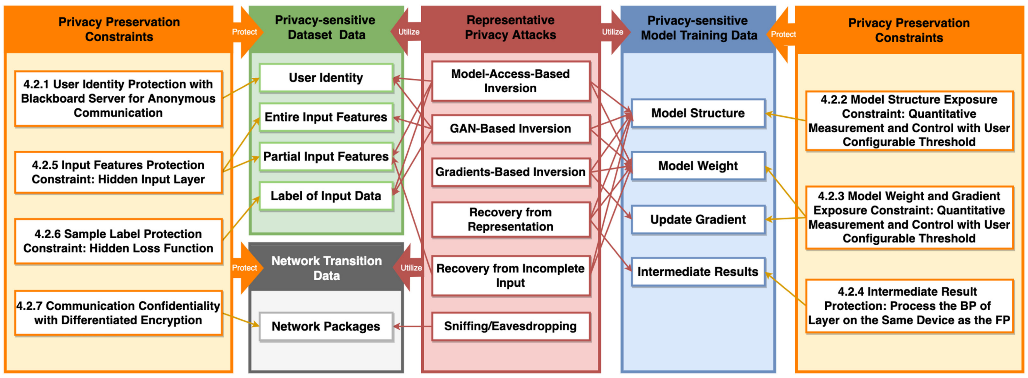
Privacy Preservation Constraints
4.2.1 User Identity Protection with Blackboard Server for Anonymous Communication
4.2.5 Input Features Protection Constraint: Hidden Input Layer
4.2.6 Sample Label Protection Constraint: Hidden Loss Function
4.2.7 Communication Confidentiality with Differentiated Encryption
Protect
Privacy-sensitive Dataset Data
User Identity
Entire Input Features
Partial Input Features
Label of Input Data
Network Transition Data
Network Packages
Utilize
Representative Privacy Attacks
Model-Access-Based Inversion
GAN-Based Inversion
Gradients-Based Inversion
Recovery from Representation
Recovery from Incomplete Input
Sniffing/Eavesdropping
Privacy-sensitive Model Training Data
Model Structure
Model Weight
Update Gradient
Intermediate Results
4.2.2 Model Structure Exposure Constraint: Quantitative Measurement and Control with User Configurable Threshold
4.2.3 Model Weight and Gradient Exposure Constraint: Quantitative Measurement and Control with User Configurable Threshold
4.2.4 Intermediate Result Protection: Process the BP of Layer on the Same Device as the FP

Fig. 2. Logic-chain of the privacy preservation mechanism: from privacy-sensitive information classes to scheduling constraints.

framework-structure objected, and the others are the quantitative constraints for the task scheduling of the global scheduler. Some of those constraints are configurable and depend on the trusting level of the users toward the edge devices, including the tradeoffs between privacy and training efficiency.

4.2.1 User Identity Protection with Blackboard Server. Since attacks like Model-access-based Inversion can abuse user identity to infer user privacy, we do not allow the edge and the terminal devices to directly communicate with each other in our proposed framework. Instead, we set up an anonymous Blackboard forwarder server between edges and terminal devices to transmit all layer-based tasks' weights, gradients, intermediate results, and description data. The Blackboard itself does not undertake any calculation work; it just caches the newly arrived tasks and forwards them at the appropriate time in an anonymous way according to the instructions of the global scheduler locates on it.

4.2.2 Model Structure Exposure Constraint. Since the user model's structure information is almost utilized by all kinds of attack methods, it must be well preserved through specific mechanisms. First, we assume that only when assigning two or more neighbor layer-tasks on the same edge device in a single round will the structure of these user model layers be exposed to this edge. In other words, in the anonymous communication scenario, if the scheduler do not assign connected layers to the edge, it will not identify the owner and relative orders of layer-tasks. On the contrary, if the scheduler assigns neighbor layers on edge, the communication of intermediate result transition between layer-tasks will no more go through the network and re-forward by the Blackboard but through the in-memory communication inside the device, which exposes the fact that the layers are connected.

We further assume that the edge will record all exposed model structure information. Based on the above assumptions, we design the following equation with a user-given threshold to measure and constrain the structure information leakage risk of each user on every single edge in a quantitative way shown in Equation (7).

$$\sum_{l \in L_m} \sum_{l' > l, l' \in L_m} \left\{ \left(\frac{1}{\mu} \right)^{l' - l} \right\} \leq \Omega_m, \forall m \in T, i \in E \tag{7}$$

The μ is a configurable hyperparameter of the model and $\mu \geq 1$. The LHS formulation measures the potential structure information leakage of each exposed layer pair l and l' of model m on device

e. While the Ω on the RHS is another configurable hyperparameter, it identifies a threshold for the corresponding risk. The primary property of this measurement is that the more these exposed layers are clustered close together, we assume that they will more easily expose the privacy. This is because if there are fewer unknown layers between the exposed layer segments, it will be easier for the attacker to predict or approximate the structure of these unknown layers, and therefore more likely to expose the privacy.

4.2.3 *Weight and Gradient Exposure Constraint.* The model weight is also almost utilized by every kind of attack method, and gradient leakage is becoming one of the most serious threats faced by the FL. Since the size of a layer's weights returned to the terminal should always equal its gradient in each training round, constraining the weight exposure can also preserve privacy of the gradient. What is more, since DL models generally have large-scale weights and generate different versions of weights in each single training round, storing these weight versions will cause inevitable efficiency and disk occupancy burdens for any edge device. Therefore, we further assume that the edge device will only be able to maintain the weight and gradient versions of each model generated in the latest time period Δt. Therefore, we design another configurable constraint on the offloaded weight amount as shown in Equation (8):

$$\sum_{k \in K} s_{w,m,l_k} \mathbb{1}\left(i_k^{asg} \neq u_k\right) \mathbb{1}\left(t_k^a \leq \hat{t} + \Delta t\right) \leq \Gamma_m, \quad \forall \hat{t} \leq t_k^a, m \in T, \tag{8}$$

where the Γ_m on the RHS is a configurable threshold that is similar to the Ω_m in Equation (7), depending on user's trust level of edges and the tradeoff between the efficiency and the privacy. With this constraint, the offloading of the weight of a user model will be limited and preserved.

4.2.4 *Intermediate Result Protection Constraint.* In the proposed framework, to avoid the inefficiency of re-computation, the edge device will cache the intermediate result generated in the FP of different layer-tasks in the local memory or disk, which is prepared for the gradient calculation in the BP. To well preserve the intermediate results from being utilized by privacy attacks, we assign the BP of each layer to the device that previously executes its FP calculations in the same round. The motivation of this constraint is that if the scheduler assigns the BP of a layer to another new edge device, then the previous device will be required to send the FP intermediate results to it, which will cause additional information leakage and raise the privacy risk.

4.2.5 *Input Features Protection Constraint.* Obviously, the entire primary features of the training samples should never be directly exposed to any untrustworthy device during the training. Since some privacy attacks further utilize the partial features to steal the user privacy, it also needs to be well preserved. In our framework, we always constrain the task of the input layer to be executed on the terminal device of its owner to avoid sending any part of the primary data sample to an external device, as shown in Equation (9).

$$i_k^{asg} = u_k, \forall k \in K \mid l_k = 1 \tag{9}$$

4.2.6 *Sample Label Protection Constraint.* In the Model-access-based and Gradient-based Inversion privacy attacks, the training sample's label is utilized to steal the privacy. To preserve the sample label, we constrain the calculation of the loss function to be executed only on the owner's terminal device, avoiding exposing the label and loss function structure, which is similar to our preservation on sample features. Here, we model the loss function as the last layer and can derive the constraint Equation (10).

$$i_k^{asg} = u_k, \forall k \in K \mid l_k = l_m \tag{10}$$

4.2.7 Communication Confidentiality. In our framework, non-terminal devices include the edge and the Blackboard server. In the worst case, it is reasonable to assume that all devices except terminal devices are untrustworthy. Therefore, we need to defend edge and Blackboard separately. Edge is responsible for calculations, therefore it must obtain each model layer's parameters, gradients, inputs, and intermediate results, but it is unnecessary for it to know the macro information as user identity and model structure. Blackboard server is responsible for anonymous forwarding and, since the scheduler relies on it, it needs to know all the macro information, including user identities and model structures, but it does not need to know the specific calculation details.

In this way, when we apply the encryption to keep the transition confidentiality from general network attacks like sniffing, we allow all terminals and edges to share the first set of public keys. The layer parameter, gradient, input, and intermediate result will be applied asymmetric encryption with this set of keys. After that, the terminal and the Blackboard will share another set of public keys for encrypting the macro information, including the device & network configuration, user identity, and model configuration. The purpose of this strategy is to effectively isolate the sensitive information held by the Blackboard and other edge devices, especially prevent the Blackboard from sniffing during its forwarding. Furthermore, some lightweight selective encryption methods [19] can be applied when the energy and time consumption of the encryption is a concern.

4.3 Privacy-aware Scheduling Optimization

Since in FL the overall training efficiency of a user model depends on all the terminal participants' completion time on their own user-side training tasks, maximizing the overall training efficiency according to the pre-defined user privacy parameters equals to minimizing the average completion time in each training round. The completion time of a user-side training round depends on the sum of the time consumption of the FP and BP of all model layer. Therefore, the optimization problem of minimizing the average completion time of each round can be further equally transferred to the problem of minimizing the average completion time of each layer-based FP or BP task during the scheduling procedure with defined privacy constraints. The completion time of each layer-based task is determined by its computation cost and the communication cost; in the following sections, we propose methods to properly measure these two kind of costs.

4.3.1 Computation Cost Measurement. First, to measure the computation cost of each task, we need to measure the workload of the primary DL layer it bases on through the layer configuration information. A possible approach applied in recent works[16] is to compute the FLOPs of the layer.

$$C_{m,l} = FLOPs\left(\tau_{m,l}^{lyr}\right) \tag{11}$$

The $C_{m,l}$ represents the floating-point operations needed to compute the FP and BP of the layer l in the terminal user's model m.

$$FLOPs\left(\tau_{m,l}^{lyr}\right) = FLOPs_F\left(\tau_{m,l}^{lyr}\right) + FLOPs_B\left(\tau_{m,l}^{lyr}\right) \tag{12}$$

More specifically, as in Equation (12), since the overall FLOPs measurement of a layer equals the sum of its FLOPs in both forward and backward propagation. The $T_{m,l}^{c}$ in Equation (13) represents the computation time cost when scheduling the device j to compute the layer-based model of layer l in user model m, while the f_j means the floating-point operation frequency of device j. To simplify the expression, we assume the f_j here to be a constant, or it can also be treated as the average operation frequency during the computation period. In the following part, we will discuss the FLOPs calculation template for some most commonly used layers in deep learning user models.

$$T_{m,l}^{c} = \frac{C_{m,l}}{f_j} \tag{13}$$

(1) FLOPs Measurement for Dense Layer. Following forms measure the FLOPs for FP and BP of a dense layer with given configuration, where $M_{m,l}$ equals to $s_{in,m,l}$, $N_{m,l}$ equals to $s_{out,m,l}$.

$$FLOPs_F\left(\tau_{m,l}^{dense}\right) = (2M_{m,l} - 1)N_{m,l} \tag{14}$$

$$FLOPs_B\left(\tau_{m,l}^{dense}\right) = (2N_{m,l} - 1)M_{m,l} + N_{m,l}M_{m,l} \tag{15}$$

(2) FLOPs Measurement for Convolutional Layer. Following two forms measure the FLOPs in FP and BP for the convolutional layer with given configuration, where K means the kernel size, C_{in} means the input channel size, C_{out} means the output channel size, H and W mean the height and width of the output tensor of this layer.

$$FLOPs_F\left(\tau_{m,l}^{conv}\right) \approx \left(2K_{m,l}^2 C_{in,m,l} - 1\right)H_{m,l}W_{m,l}C_{out,m,l} \tag{16}$$

$$FLOPs_B\left(\tau_{m,l}^{conv}\right) \approx C_{out,m,l}C_{in,m,l}K_{m,l}^2(2H_{m,l}W_{m,l} - 1) + (2C_{out,m,l} - 1)C_{in,m,l}K_{m,l}^2 H_{m,l}W_{m,l} \tag{17}$$

Since the Pooling Layer can also be considered as a special kind of convolutional layer, the FLOPs measurement for Pooling Layer can also use the above two formulations. And the FLOPs measurement for activation layers depends on which activation function is used, which needs to be further verified and discussed.

4.3.2 Communication Cost Measurement. Furthermore, we also need to measure the communication delay according to different cases, which can be basically classified into two different classes: FP phase communication and BP phase communication.

Forward Propagation Communication Measurement. In the FP of a layer l in model m, the transition of the layer's input and weight before the computation procedure will also cause corresponding delays. The delay differs by the situation, depends on the type of devices where the layer $l-1$ and l lays. In the following formulations, we represent the device assigned with layer $l-1$ as device i, the device chosen to process the layer l as the device j. $F_{m,l}$ means the communication delay of layer l in model m during its FP. The owner terminal of the task is device u.

Case 1: $i \in T$ and $j \in T$. In this case, we must have $i = j = u$, since $D_{m,l} = u$. This case includes the subcase where the $l = 0$, since the FP of the input layer of any model will be fixedly assigned to terminal u because of the privacy constraint proposed in the previous section. Then, we intuitively have the $F_{m,l}^w = 0$ and $F_{m,l}^{in} = 0$, and we have the real communication delay here to be 0.

$$F_{m,l}^{t\to t} = F_{m,0} = 0 \tag{18}$$

Case 2: $i \in E$ and $j \in T$. In this case, the real communication delay should be totally caused by its input. It is worthwhile to note here that similar to the modeling of computation delays, the p here should have been a random variable, but to simplify the expression, we assume it to be a constant. It can also be treated as the average network bandwidth during the transition period. It is same in the following few formulations. We also assume that only the device has the constraint on bandwidth, while the transition bandwidth in the network is always large enough to avoid the message blocking.

$$F_{m,l}^{t\to e} = F_{m,l}^{in} = \frac{s_{m,l}^{in}}{p_i} + \frac{s_{m,l}^{in}}{p_j} \tag{19}$$

Since the terminal device $j = u$ has all the weights for the model training, we intuitively have the weight delay $F_{m,l}^w = 0$. The only communication delay will be caused by the transition of previous layer output from device i to device j.

Case 3: $i \in T$ and $j \in E$. In this case, we must have $i = u$, and the communication delay of the weights an be calculated as follows:

$$F^{w}_{m,l} = \frac{s^{w}_{m,l}}{p_u} + \frac{s^{w}_{m,l}}{p_j}. \tag{20}$$

The communication delay of the layer input should be similarly measured same as in the Case 2. Since in this case the terminal device needs to sequentially send both the weights of layer l and the output of the layer $l-1$ as the input of the layer l, we have the expression as follows:

$$F^{t \to e}_{m,l} = F^{w}_{m,l} + F^{in}_{m,l}. \tag{21}$$

Case 4: $i \in E$ and $j \in E$, with $i = j$. In this case, there will be no delay for the network transition of the output of layer $l-1$ to layer l. Since they are allocated on the same device, the data transition will only happen in the memory of the device and it has a relatively low cost, which can be omitted in our modeling. However, edge device needs new weights for layer $l+1$ to continuously calculate the task. Therefore, in the FP procedure it still requires the terminal u to give the weights of layer $l+1$ to device i, then in this case the real communication cost will include and only include the $F^{w}_{m,l}$, which can be measured with the same expression as shown in the Case 3.

$$F^{e \to e}_{m,l} = F^{w}_{m,l} \tag{22}$$

Case 5: $i \in E$ and $j \in E$, with $i \neq j$. In this case, the calculation of the real communication delay $F^{e \to e'}_{m,l}$ should be different from the discussion for the case where $i \in E$ and $j \in E$ and $i = j$. Since in this case the terminal device u and edge device i can simultaneously send the data to the device j, the real delay of the communication for task of layer l in this case should be the larger one of the weights transition and the last layer's output transition, while the $F^{w}_{m,l}$ and $F^{in}_{m,l}$ can be measured in the same way as in previous cases.

$$F^{t \to e}_{m,l} = \max\left(F^{w}_{m,l}, F^{in}_{m,l}\right) \tag{23}$$

Backward Propagation Communication Measurement. In the BP process of a layer, the generation of the gradients needs to wait for the transition of the gradient of its output from the next layer. The specific communication delay also needs to be further discussed by cases.

In following formulations, we represent the device chosen to process the BP of layer l as the device i, the device assigned with layer $l+1$ as device j. While $B_{m,l}$ means the real communication delay of layer l of model m in the current round in the BP phase. u stands for the owner terminal device.

Case 1: $i \in T$ and $j \in T$, or $i \in E$ and $j \in E$ with $i = j$. When the adjacent two layers are allocated on a same device, the transition will no more go through the Blackboard or network. The only delay will be caused by in-memory operations, which could be omitted in our framework modeling.

$$B^{e \leftarrow e}_{m,l} = B^{t \leftarrow t}_{m,l} = 0 \tag{24}$$

Case 2: $i \in E$ and $j \in T$. In this case, the real communication delay should be the transition time of previous layer's output.

$$B^{e \leftarrow t}_{m,l} = \frac{s^{out}_{m,l}}{p_j} + \frac{s^{out}_{m,l}}{p_i} \tag{25}$$

It is worthwhile mentioning that although the gradients on layer $l+1$'s weights generated on device j during the BP also need to be sent by the device j to its owner terminal device u, we can omit the transition delay of the gradients of $l+1$'s weights when we are calculating the actual communication delay of the task of layer l on device i. That is because the procedure of layer l's

BP only depends on the gradients of its output and has no direct dependency with the gradients of layer $l+1$'s weights.

Therefore, in our framework, we set the device j to always first pass the gradients of its input back to the device i where the previous layer l is located after completing the gradient calculation of the input and parameters of its layer $l+1$. After that, it will pass the gradient of its weights back to the terminal u. It is also assumed that the transition time of the gradients of the weights does not affect other operations in the framework.

Case 3: $i \in T$ and $j \in E$, or $i \in E$ and $j \in E$ with $i \neq j$. In this case, the actual communication delay will only be caused by the input gradients transition delay caused by the edge j.

$$B_{m,l}^{e \leftarrow e'} = B_{m,l}^{t \leftarrow e} = \frac{s_{m,l}^{out}}{p_j} + \frac{s_{m,l}^{out}}{p_i} \tag{26}$$

It is same as the previous case where $i \in E$ and $j \in T$ on the mathematical perspective.

4.3.3 Optimization Problem Formulation. First, we use the following formulation and $T_{m,l}^{w}$ to summarize the real communication delay of layer l and model m in different cases:

$$T_{m,l}^{w} = \begin{cases} F_{m,l}^{x \rightarrow y}, \ if\ in\ FP \\ B_{m,l}^{x \leftarrow y}, \ if\ in\ BP \end{cases} \quad x, y \in E \cap T. \tag{27}$$

After the declaration of both computation and communication delays brought by different layer-based tasks, we are now going to introduce the concept of the task completion time. The completion time of a task does not simply equal to the sum of both its communication delay and computation delay. Instead, it counts for the time interval from the arrival time of the task to its eventual completion. The completion time helps to represent the total waiting time of the task publishers or the users for the training results in a round. Whenever a new task arrives on the central Blackboard server, the global scheduler will cache it into the memory. At any possible time point, the scheduler always has two possible actions for the task cached in the memory: one is to publish the task to the next device, which may be either idle or has a task queue. Another one is to keep it in the memory and wait to publish it to a device later. In this way, the completion time will always be equal or larger than the sum of the communication delay and computation delay and can be defined as in Equation (28).

$$T_{k,i}^{C} = T_{k}^{B} + \max\left(T_{k,i}^{Q}, T_{k,i}^{w}\right) + T_{k,i}^{c} \tag{28}$$

In Equation (28), Blocking Time T^B equals to the time interval between the initial arrival time of the task and the time point scheduler publishes this task to the next device. More specifically, we have $T^B = t^{pub} - t^a \geq 0$. The Queuing Time $T_{k,j}^{Q}$ means the time of task k queuing on its executor device j before the actual execution. It equals to the sum of the computation delay of all the previously arrived or executing tasks before the arrival time of task k on j. Here, we define U_t as the set of tasks that has arrived but is unfinished at time t. To express $T_{k,j}^{Q}$, we can first define another set $Q_{k,t^{pub}}$ as the set of all tasks previously queued on the target device before the arrival of task k on it, which should be first executed before the execution of task k.

$$Q_{k,t^{pub}} = \left\{\hat{k} \in U_{t^{pub}} | j_{\hat{k}} = j_k, t_{\hat{k}}^{pub} + T_{\hat{k},\hat{j}}^{w} \leq t_{k}^{pub} + T_{k,j}^{w}\right\} \tag{29}$$

Then, we use the $Q_{k,t^{pub}}$ to define the Queuing Time $T_{k,j}^{Q}$ for k in Equation (30).

$$T_{k,j}^{Q} = \sum_{\hat{k} \in Q_{k,t^{pub}}} \left\{T_{\hat{k},j}^{c}\right\} \tag{30}$$

Since the transition of the computational dependent data, including the weights, inputs, and gradients of layer l and the device j's execution of its task queue, happens simultaneously, the upper bound of these two will decide the delay. In this way, we use the max in Equation (28).

As we have discussed in previous sections, the completion time of the entire user-side training round includes both the FP and BP of a model, and it further decides the overall training efficiency of the FL model. In this way, for our formal scheduling optimization problem, maximizing the overall training efficiency with pre-defined privacy constraints equals to minimizing the completion time of the entire single round training on the user-side for each $\hat{u} \in T$ and we have the optimization problem as follows:

$$\begin{aligned} \min_{\boldsymbol{t^{pub}}, \boldsymbol{j}} \quad & \mathbb{E}_{\hat{u}} \left[\sum_{k | l_k \in L_m} \mathbb{E}_{f_{\hat{u}}, p_{\hat{u}}} \left[T^C_{k,j} | u_k \in \hat{u} \right] \right] \\ s.t. \quad & t^{pub}_k \geq t^a_k, \\ & j_k \in D^{t^{pub}_k}_{u_k, l_k}, \\ & (7), (8), (9), (10). \end{aligned}$$

The optimization variables are vector $\boldsymbol{t^{pub}}$ and vector $\boldsymbol{j}$; both of them share the same size as set K. The $\boldsymbol{t^{pub}}$ is scheduler's publish time of tasks, and $\boldsymbol{j}$ is the decided executor of tasks. Since the term $T^C_{k,j}$ depends on random variables including p and f and the arrival time of tasks always depends on its previous task's completion time, which is also stochastic, we use an expectation to summarize the randomness. The constraints in the problem ensure the integrity of the optimization and the user privacy. Equations (7) and (8) are both configurable and consider the efficiency-privacy tradeoff. Since minimizing the overall time consumption of FP and BP equals to minimizing the expected completion time of each layer-task, this optimization problem can be equivalently transferred into another simpler form as follows:

$$\begin{aligned} \min_{\boldsymbol{t^{pub}}, \boldsymbol{j}} \quad & \mathbb{E}_{f_{\hat{u}}, p_{\hat{u}}, k} \left[T^C_{k,j} \right] \\ s.t. \quad & t^{pub}_k \geq t^a_k, \\ & j_k \in D^{t^{pub}_k}_{u_k, l_k}, \\ & (7), (8), (9), (10). \end{aligned}$$

5 ONLINE SCHEDULING ALGORITHM

Even if we assume that the final optimization problem modeled in our Section 4 does not have the challenges caused by the randomness of the environment, which is to assume that the computing power and network condition of all devices are constant instead of changing with time, this optimization problem will still be an NP-hard problem. Therefore, we have to find an approximation algorithm to achieve efficient enough scheduling decisions. At the same time, we hope that the approximation scheduling algorithm has the properties of the online algorithm, which can continuously eliminate the impact brought by the environment randomness on the entire scheduling during the scheduling procedure, to make sure the scheduling algorithm is still effective in minimizing the average waiting time of tasks.

We mainly refer to the offline task scheduling algorithm on the multiple edges situation proposed by Xiaolin Fang et al. [2] and improve it to be an online algorithm that fits our optimization problem. The significant differences between the proposed online algorithm and the original offline algorithm are: (1) The online optimization process will be called multiple times during the

ALGORITHM 1: Virtual Machine Speed

```
Require: Given $T_k$, $E_k$ and their configuration tuples, task $k$ and its publish time $t_0$, and the published but
    not finished tasks set $K_p$.
Ensure: $s(k,t)$, execution speed of $k$ in the proposed VM at any time $t > t_0$.
1: Initialize $s(k, t_0) = 0$;
2: for any timestamp $t > t_0$ do
3:     $s(k,t) = s(k,t-1)$;
4:     for each device $i \in T_k \cup E_k$ do
5:         if $t = \max\{T^Q_{k,i}, T^w_{k,i}\} + t_0$ then
6:             $s(k,t) = s(k,t) + \mathbb{E}[s_i]$;
7:         end if
8:     end for
9: end for
```

ALGORITHM 2: Near Future Task Arrival Prediction

```
Require: Given $T$, $E$, and their configuration tuples, $K_p$ and current time $t_0$.
Ensure: $C_p$, predicted tasks will arrive in the near future.
1: for $k \in K_p$ do
2:     $t_k = t_0 + \max\{\mathbb{E}[T^Q_k], \mathbb{E}[T^w_k]\} + \mathbb{E}[T^c_k] - \hat{T}^p_{k,t_0}$;
3:     Build expected new task $k'$;
4:     Set $t^a_{k'} = t_k$;
5:     $C_p.append(k')$;
6: end for
```

scheduling process (after each Trigger Time) to update the scheduler's knowledge about the current environment and task-queuing situations, and eliminate the gap between the predictive estimation and the reality emerged after the last trigger time. (2) In each trigger time, the scheduler will predict the specific arrival time and workload of new tasks that will arrive soon, based on which the scheduler can process the local optimization.

Similar to the original offline algorithm [2], we first build a **Virtual Machine (VM)** to summarize computation and communication capabilities of all devices as shown in Algorithm 1. However, to enable our scheduler to process the optimization in any timestamp during the whole scheduling procedure, we no more assume that the devices are initially idle in our proposed VM. Instead, we consider the device to be either idle or executing on a task queue. The T_k in the input is the subset of terminals allowed to execute the tasks from the owner of the task k, having $T_k = u$. The E_k means the edges that pass the privacy constraints of k's owner can assist the computation of k.

What is more, a prediction method for the coming tasks in the near future is proposed in Algorithm 2. The K_p in the input represents the set of tasks that have been published but not yet finished. The symbol $\hat{T}^p_{k,t_0}$ in Line 2 is defined as the time interval between the publication of task k and time t_0, which is a determinant value. The algorithm will output a set of predicted tasks based on current tasks' transition and execution situations.

Algorithm 3 is the main part of the optimization, whose Lines 1 to 17 is the preemptive scheduling bases on the task arrival prediction and defined VM, while from Lines 18 to 25 is the eventual non-preemptive version bases on the preemptive completion list $\mathbb{L}$ achieved in the preemptive part.

In preemptive scheduling, the scheduler assumes our non-preemptive layer-based FL tasks as preemptive ones and always executes the one with the earliest completion time on the defined VM at any timestamp. Then the completion list $\mathbb{L}$ sorts these tasks in the order of their predicted completion time. After that, the scheduler publishes the arrived tasks in the given order in $\mathbb{L}$ to

ALGORITHM 3: Online Scheduling Algorithm

```
Require: Given $T$, $E$, and their configuration tuples, all tasks set $K$, current trigger time $t_0$, and the tasks set
    stored in Blackboard's buffer $B$.
Ensure: Online scheduling actions at time $t_0$.
 1: $C_p = NearFutureTasks(T, E, K_p \in K, t_0)$;
 2: $\bar{C} = \{B\}, \mathbb{L} = list[]$;
 3: for time $\bar{t}$ with new $\hat{k} \in C_p$ arriving do
 4:     Calculate the remaining $FLOPs$ of last executed task $k^{old}$;
 5:     if expected left FLOPs of $k^{old} = 0$ then
 6:         $\bar{C}.dequeue(k^{old})$;
 7:         $\mathbb{L}.append(k^{old})$;
 8:     end if
 9:     $\bar{C}.append(\hat{k})$;
10:     $min = +\infty, k^{new} = 0$;
11:     for $\bar{k} \in \bar{C}$ do
12:         Calculate the expected completion time $\mathbb{E}[c_{\bar{k}}]$ of task $\bar{k}$:
            $\mathbb{E}[c_{\bar{k}}] = \arg\min_{t'} \{\sum_{t' \geq \bar{t}} s(\bar{k}, t') \leq \bar{k}.FLOPs\}$
13:         if $min \geq \mathbb{E}[c_{\bar{k}}]$ then
14:             $min = \mathbb{E}[c_{\bar{k}}], k^{new} = \bar{k}$;
15:         end if
16:     end for
17: end for
18: for $\hat{k} \in \mathbb{L}$ do
19:     if $\hat{k} \in B$ then
20:         $i = \arg\min_{\hat{i}} \mathbb{E}[T^C_{k,\hat{i}}], i \in \{T_k \cap E_k\}$;
21:         Publish task $\hat{k}$ to device $i$;
22:     else
23:         break;
24:     end if
25: end for
```

the device, which can finish it earliest, until it finds that the next task is from the predicted task set K_p. It means that the next task to publish has not arrived in the buffer yet; waiting for its arrival and publishing it in a later timestamp is a better scheduling action and achieves less average completion time. Therefore, in each trigger time, the scheduler will publish all the arrived tasks in buffer B before the first predicted task in K_p in the order of $\mathbb{L}$. One of the properties of the proposed algorithm is its general preference to first publish and execute the smaller tasks with lower computation and communication costs. Similar to some operating systems' giving priority to scheduling processes with smaller workloads, the intuition behind this is relatively straightforward: This will generally help to minimize the average completion time of tasks, which is the objective of our optimization problem as shown at the end of Section 4.3.3. If we publish and execute the large task first, then all other smaller tasks will need to queue and wait for its completion before their execution; while when smaller tasks can be executed first, on the contrary, the completion time of most tasks can be reduced. However, this may also lead to the resource starvation problem for complex models' tasks. We believe that this potential problem can be avoided by further applying modifications to the algorithm: (1) Set an initial priority for each task according to their expected complete time, which will increase with time, and the scheduler will consider publishing tasks with higher priority earlier; (2) Set a maximal time a task can stay in the Blackboard buffer. After reaching this threshold, the scheduler will have to publish this task immediately.

We define the trigger time of the scheduler as all timestamps where the local scheduling action may change, which is generally each time when a new task arrives in the Blackboard and the very initial time when all devices are idle. In theory, our proposed online version of the multiple edge scheduling algorithm will approach the approximation ratio of the original offline version, which is $2 + \frac{\bar{f}^{max}}{\bar{f}^{min}}$ without considering the back transition time of results, where $\bar{f}^{max} = \max_i^{T \cap E} \bar{f}_i$ and $\bar{f}^{min} = \min_i^{T \cap E} \bar{f}_i$. The specific approximation ratio depends on the bias and the variance of probability distribution estimations of each f_i and p_i.

6 FRAMEWORK EVALUATION

This section will evaluate the properties of the proposed training framework and online scheduling algorithm through a series of simulation experiments. After briefly introducing experiment settings, we will validate the efficiency improvement of the framework, observe tradeoffs between the training efficiency and different privacy constraints, the effect of the ratio between the number of edges and terminals on the training, and visualize how the change on privacy constraints influence the scheduler's decision.

6.1 Experiment Setting

To simulate the environment in which the framework is applied, we first simulate the computing power and communication bandwidth of each edge and terminal device with different Gaussian distributions that conform to the real-world situation. Considering some special processing units like GPU could be installed on the edges, the mean of computing frequencies $\bar{f}$ of edge devices is generally about 4 to 8 times that of terminal devices. For the network quality, since we assume the terminals generally communicate with the Blackboard server with the wireless network while a wired network can connect the edges and the Blackboard, edges are allocated with about 30%–40% more bandwidth than terminals. For the simulation of different user models, we decompose four representative DL models, including LeNet, AlexNet, ResNet18, and VGG16, into sets of layer-based components. Since the structure complexity of the model differs from each other, the number of the eventual decomposed components varies from 12 to 60.

6.2 Training Efficiency Improvement

To validate the effectiveness of our framework and online scheduling algorithm, we compare the average model training efficiency in the following four situations: (1) pure terminal case without the assistance of edges, (2) edge-assisted FL with a random scheduling algorithm, (3) edge-assisted FL with the proposed online scheduling algorithm, and (4) edge-assisted FL with the online algorithm but without privacy constraints. In cases (2) and (3), the system configures the privacy constraints to be very rigorous to ensure the simulation is conducted under the premise that all users have high privacy awareness and least trust on edges. All users' structure constraint threshold Ω is configured to be 0, which means the scheduler does not allow any exposure to the model structure. We set Γ to be 0.05, meaning that at most 1/20 weights could be offloaded from the terminal to the edges for every user within the identified time window. We apply the test set for four different learning models, including the LeNet, AlexNet, ResNet18, and VGG16, with three terminals and two edges in the environment. The red bars in Figure 3 represent the original average training time costs with only terminals' participation and are fixed to be 1 for each model. The blue bars represent the average time cost ratio relative to the previous case, with edge's assistance but a trivial scheduling algorithm. The green bars represent the time ratio with the proposed online algorithm with strict privacy constraints, and the orange bars presume the complete trust in edge devices or without any privacy concern, therefore considering no privacy constraint.

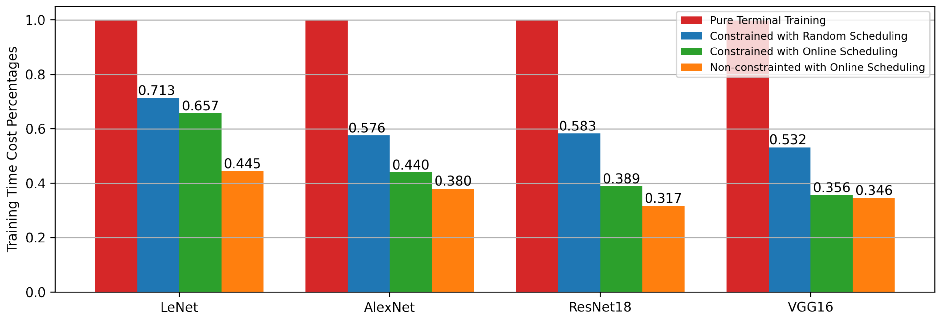

Fig. 3. Comparing the average training cost of four different representative DL models with four different devices, scheduling, and privacy settings.

From the data in Figure 3, we can see that for LeNet, the training efficiency improved about $1/0.713 - 1 = 40\%$ when introducing edge assistance. And the efficiency improves 52% when further applying the proposed online algorithm. The efficiency can be optimally improved 125% when edges can be fully trusted and without any privacy constraint. The improvement becomes more obvious when the model structure and weight scale grow to be more complex. For example, in VGG16, the improvements on the efficiency of cases (2) to (4) compared to case (1) are 92%, 180%, and 189%. And the efficiency of constrained online algorithm case will also more approach the fully trusted case.

6.3 Tradeoff on Model Structure Constraint

In the proposed framework, we assume that the user's requirements on privacy preservation depend on both the user's trust level in the external devices and the privacy sensitivity of the training data. We find that there naturally exists a tradeoff between training efficiency and user privacy. Since privacy requirements vary from user to user, we should allow some users to trade partial privacy for efficiency within a reasonable range. Therefore, we set the threshold in Privacy Constraint Equations (7) and (8) as hyperparameters that each user can configure according to specific situations.

In this way, we conducted simulation-based explorations on four different representative DL models in a relatively simple device scenario, as shown in Figure 4. Model types include LeNet, AlexNet, ResNet18, and VGG16. From the figure, we can see that for the LeNet case, when configuring the threshold Ω from 0 to 0.003, which is making the constraint less strict, the performance increased by about 7%. For AlexNet, ResNet18, and VGG16, the performance, respectively, increased 5%, 9%, and 10%. We find that, in our framework, with the increasing privacy preservation on the model structure, the decreasing performance is relatively less obvious. To better explain this phenomenon, we make a set of detailed visualization of the scheduling procedure in Section 6.6.

At the same time, this result can serve as a reference when users configure their privacy hyperparameters: Since the protection of the model structure has very limited discounts on the efficiency, when we need to further enhance privacy protection, we can give priority to reducing Ω. At the same time, relaxing this constraint will not bring obvious improvement on the efficiency.

6.4 Tradeoff on Model Weight & Gradient Constraint

Similar to the constraint on the model structure, there also exists a tradeoff between the training efficiency and the privacy constraint on the model weight and gradient. We conducted four sets of

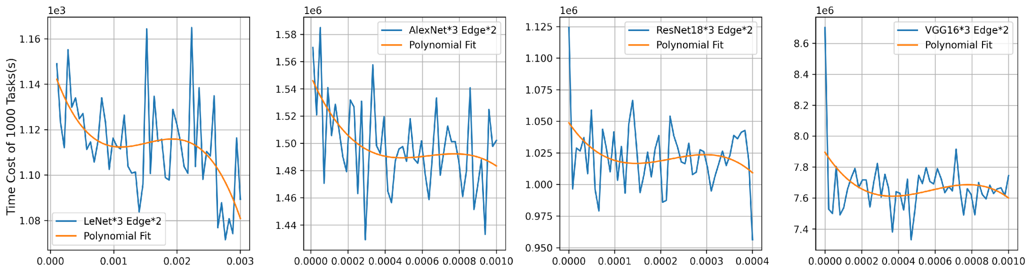

Fig. 4. The average time cost of completing 1,000 layer-tasks with different privacy thresholds Ω with polynomial fitting, from which we can find out the tradeoff between performance and privacy preservation on the model's structure.

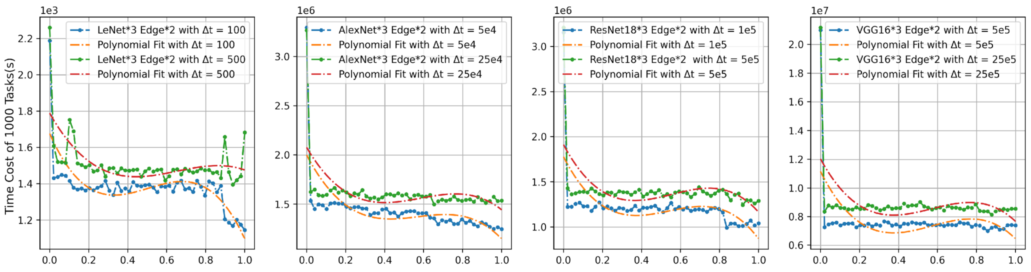

Fig. 5. The average time cost of completing 1,000 layer-tasks with different privacy thresholds Γ and Δt, with polynomial fitting, from which we can find out the tradeoff between performance and privacy preservation on the model's weights and gradients.

experiments, respectively, on LeNet, AlexNet, ResNet18, and VGG16, to see the effect of the configurable threshold Γ and hyperparameter Δt in constraint Equation (8) on the training efficiency. In the experiment, as shown in Figure 5, we set different parameters Δt for different models, which roughly equals to 1/10 and 1/2 of their time cost of training 1,000 tasks. We set Δt equals to 100 & 500 for LeNet, 50,000 & 250,000 for AlexNet, 100,000 & 500,000 for ResNet18, and 500,000 & 2,500,000 for VGG16. We change the weight & gradient exposure threshold Γ from 0.0 to 1.0 in each case, from the most rigorous to the most relaxed. We can observe that the model-training efficiency will experience significant increments when the constraint is being relaxed. To better express the trend, a set of polynomial fitting curves is provided with a degree of 3. For the LeNet set, the training efficiency increased about 57%–91% for different Δt. For the AlexNet and VGG16, it ranges from 112% to 188%. For the ResNet18, it is about 140%–202%. When we are setting the Γ to be 0.0, we are actually prohibiting the edges from assisting the training procedure, because we do not allow any layer's weight to be exposed in any single round. Therefore, the training efficiency at $\Gamma = 0.0$ also represents the efficiency of the traditional FL training with only terminal devices.

By further observing the characteristics of the time cost curves, we can find a similarity shared by all models and time window settings: The significant improvement of the efficiency happens in the range of $\Gamma = [0, 0.1]$. For LeNet with $\Delta t = 100$, the efficiency improved 52% and remains unchanged until about $\Gamma = 0.8$. For AlexNet and ResNet18, the efficiency improves more than about 120% and remains in the range $[0.1, 0.8]$, while for VGG16, the efficiency improved about 140% in the range $[0, 0.1]$ and remains unchanged in the following until $\Gamma = 1.0$, which actually no more constrains the weight & gradient exposure.

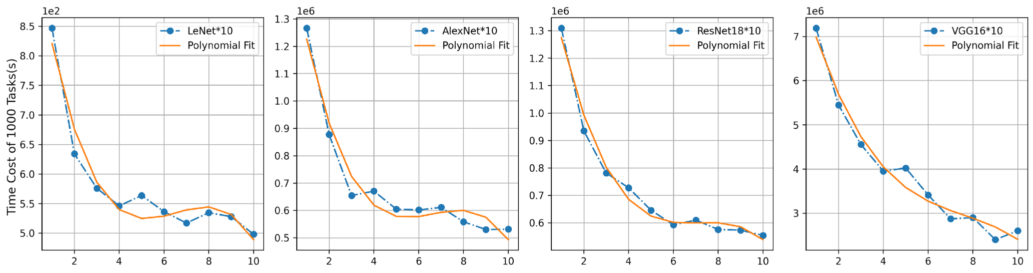

Fig. 6. The average time cost of completing 1,000 layer-tasks with different ratio of terminal number and edge number with polynomial fitting.

The result we derive from this set of experiments can also provide the users a critical reference when considering the tradeoff between the privacy constraint on model weights and the training efficiency: Since in the range of $\Gamma \in [0.1, 0.8]$, the change in the constraint threshold will not obviously affect the training efficiency, users can set a relatively more strict threshold to protect their privacy; while setting a $\Gamma \leq 0.1$ could significantly damage the efficiency.

6.5 Ratio of Edge and Terminal Number

To explore more on optimization of the edge computing resources allocation, we have conducted a set of experiments to see the influence of the ratio of the edge device number to the terminal device number on the FL training efficiency of the four models: LeNet, AlexNet, ResNet18, and VGG16. Notably, from Figure 6, we can see that when the number of the terminal is fixed to be 10 and their task amount remains unchanged, the overall training efficiency and the number of the edge do not show a linear relationship as expected. On the contrary, the efficiency increment brought by each additional edge device gradually decreases, and in some cases, the efficiency begins to remain almost unchanged after the edge number increases to a certain amount.

Actually, the improvement brought by the first few edges is also much smaller than expected. The increment on efficiency for four models when adding the edge and terminal number ratio from 0.1 to 0.2 are about 31%, 44%, 37%, and 29%, which are generally significantly smaller than the expected 50%. While for the case of LeNet and ResNet, the increment of the efficiency from increasing the ratio of edge and terminal number basically disappears after the sixth and eighth edge, respectively. Basically, for all four sets of experiments, the efficiency improvement generally reaches about 100% when adding the sixth edge into the environment.

All these provide a reference for the deployment of edge resources: With a set of edges and terminals in the proposed framework, the deployment number of edges should be based on the available number of nearby terminals, and it should be noted that, when optimizing the resources utilization, the efficiency increment each new edge can bring could be limited with the increasing number of the existing edges.

6.6 Visual Characteristics of Scheduling

To further study how the privacy constraints influence the scheduling strategy of the global scheduler, especially the model weight constraint, we have visualized the task scheduling procedure as Figures 7 and 8 in the application scenario with three terminal users training the AlexNet assisted by two extra edge devices. We use three different colors in both figures to represent the training task set, respectively, belonging to three terminal users, and each rectangular color block represents the computation procedure of a task from its beginning to the end. The row position of each rectangular color block indicates which device this task is scheduled to by the scheduler for

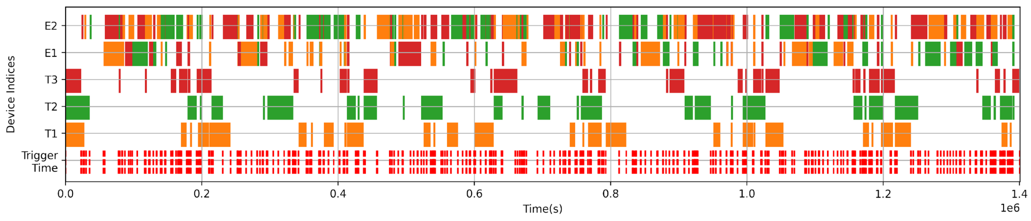

Fig. 7. Visualization of the scheduling procedure with less rigorous model structure preservation constraint, setting $\Omega = 0.003$.

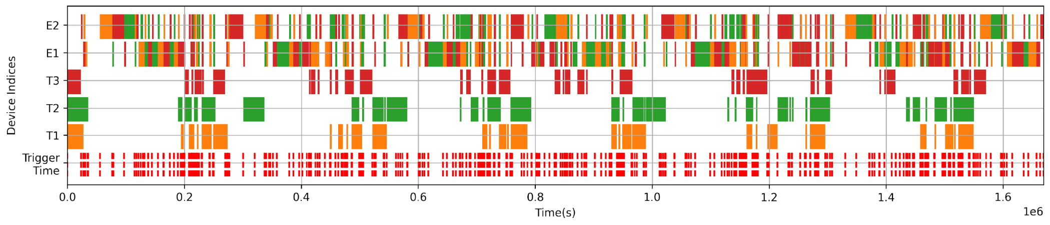

Fig. 8. Visualization of the scheduling procedure with the most rigorous model structure preservation constraint, setting $\Omega = 0$.

training. The left side of the color block represents the time when the task starts to execute on the corresponding device, while the right side represents its time to be completed on it. Therefore, the width of each color block represents the total execution time cost T_k^c of each task. We can see there are densely arranged dotted lines at the bottom of the figures, which indicate the trigger time for the global scheduler to be activated and do the local re-optimization for the scheduling. We can see that each trigger time of the global scheduler is generally after a previous task is completed, which also means a new task emerged to the scheduler in our framework, rather than evenly spaced and evenly distributed in the timeline.

As stipulated in algorithms, tasks belonging to each user can either be executed on the publisher's terminal or on edge devices that pass the privacy constraints and cannot be offloaded to other terminal devices. On the edge devices, however, tasks issued by various terminal devices are often gathered according to the suggestion of the scheduler, which aims to optimize the training efficiency. For example, we can see that there are only Orange task color patches on Terminal 1, while edge device 1 has Red, Green, and Orange color patches from three terminals. It is worth noting that, because the computing power of the edge device is significantly stronger than that of the terminal device in our experimental design, the average width of the color block deployed on the edge device is obviously much lower than that of the terminal device.

Figures 7 and 8, respectively, reflect the difference between the scheduler's scheduling pattern under different privacy constraints. Γ of both models are set to 0.5, T is set to 50,000, and the Ω in Figure 7 is set to 0.003, which is a relatively non-strict threshold for AlexNet, which allows some model structures to be exposed. While in Figure 8, Ω is set to 0, which means that the terminal users strictly do not allow any user model structure to be exposed. From the figures, we can observe that when the model structure constraint is less rigorous, tasks with originally adjacent layers belonging to the same terminal user are often assigned to the same edge device. The reason for the scheduler to decide this is to improve the overall training efficiency by reducing the communication overhead of intermediate results transition between layers. In Figure 8, we found an interesting phenomenon: Task blocks with the same color are often interleaved in two terminal devices. Strictly speaking, there are no more adjacent color blocks assigned on the same edge device. Instead, on the same edge, there will be at least one task block between two with the same color to be assigned on the terminal or another edge device. The purpose of this behavior of the scheduler is obvious: to avoid exposing the continuous layer structure to any edge by adding a small acceptable amount of communication overhead to keep making full utilization of edge computing resources while passing the model structure privacy constraint.

7 CONCLUSION AND DISCUSSION

This article proposes an energy-efficient and privacy-aware decomposition framework for the FL in the general MEC scenario and has mathematically modeled the task migrating optimization problem with specific privacy constraints. We have also proposed a privacy-aware online scheduling algorithm that can eliminate the gap between the scheduler's predictive estimation and the actual system condition, which helps solve the optimization problem. Eventually, we conduct a series of simulations and experiments to prove the efficiency improvement of the proposed FL decomposition framework. There are several aspects worth further study: (1) Further validation on the privacy preservation of the proposed decomposition framework should be conducted by implementing the actual scalable training system and applying various modern privacy attacks on it. In this way, the privacy preservation ability could be fully proven with the failure of these attacks. (2) Completing the computation and communication measurements on more DL models with higher structure complexity, whose layers may be ordered in a non-sequential way, such as the RNN models.

REFERENCES

[1] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. 2021. DAPPLE: A pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 431–445.

[2] Xiaolin Fang, Zhipeng Cai, Wenyi Tang, Guangchun Luo, Junzhou Luo, Ran Bi, and Hong Gao. 2020. Job scheduling to minimize total completion time on multiple edge servers. *IEEE Trans. Netw. Sci. Eng.* 7, 4 (2020), 2245–2255. DOI: https://doi.org/10.1109/TNSE.2019.2958281

[3] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1322–1333.

[4] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. 2014. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security'14)*. 17–32.

[5] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. PipeDream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377* (2018).

[6] Chaoyang He, Murali Annavaram, and Salman Avestimehr. 2020. Group knowledge transfer: Federated learning of large CNNs at the edge. *arXiv preprint arXiv:2007.14513* (2020).

[7] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: Information leakage from collaborative deep learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 603–618.

[8] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, et al. 2019. GPipe: Efficient training of giant neural networks using pipeline parallelism. *Adv. Neural Inf. Process. Syst.* 32 (2019), 103–112.

[9] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K. Leung, and Leandros Tassiulas. 2019. Model pruning enables efficient federated learning on edge devices. *arXiv preprint arXiv:1909.12326* (2019).

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[11] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. 2021. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Trans. Knowl. Data Eng.* (2021), 1–1. DOI: 10.1109/TKDE.2021.3124599

[12] Ruqian Lu, Xiaolong Jin, Songmao Zhang, Meikang Qiu, and Xindong Wu. 2018. A study on big knowledge and its engineering issues. *IEEE Trans. Knowl. Data Eng.* 31, 9 (2018), 1630–1644.

[13] Aravindh Mahendran and Andrea Vedaldi. 2016. Visualizing deep convolutional neural networks using natural pre-images. *Int. J. Comput. Vis.* 120, 3 (2016), 233–255.

[14] Yunlong Mao, Shanhe Yi, Qun Li, Jinghao Feng, Fengyuan Xu, and Sheng Zhong. 2018. A privacy-preserving deep learning approach for face recognition with edge computing. In *Proceedings of the USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18)*. 1–6.

[15] Jed Mills, Jia Hu, and Geyong Min. 2019. Communication-efficient federated learning for wireless edge intelligence in IoT. *IEEE Internet Things J.* 7, 7 (2019), 5986–5994.

[16] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440* (2016).

[17] Takayuki Nishio and Ryo Yonetani. 2019. Client selection for federated learning with heterogeneous resources in mobile edge. In *Proceedings of the IEEE International Conference on Communications (ICC'19)*. IEEE, 1–7.

[18] Seyed Ali Osia, Ali Shahin Shamsabadi, Sina Sajadmanesh, Ali Taheri, Kleomenis Katevas, Hamid R. Rabiee, Nicholas D. Lane, and Hamed Haddadi. 2020. A hybrid deep learning architecture for privacy-preserving mobile analytics. *IEEE Internet Things J.* 7, 5 (2020), 4505–4518.

[19] Han Qiu, Meikang Qiu, and Zhihui Lu. 2020. Selective encryption on ECG data in body sensor network based on supervised machine learning. *Inf. Fusion* 55 (2020), 59–67.

[20] Yimin Shi, Haihan Duan, Yuanfang Chi, Keke Gai, and Wei Cai. 2020. Edge-assisted federated learning: An empirical study from software decomposition perspective. In *Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 200–214.

[21] Nguyen H. Tran, Wei Bao, Albert Zomaya, Minh N. H. Nguyen, and Choong Seon Hong. 2019. Federated learning over wireless networks: Optimization model design and analysis. In *Proceedings of the IEEE Conference on Computer Communications*. IEEE, 1387–1395.

[22] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S. Yu. 2018. Not just privacy: Improving performance of private deep learning in mobile cloud. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2407–2416.

[23] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. 2019. Adaptive federated learning in resource constrained edge computing systems. *IEEE J. Select. Areas. Commun.* 37, 6 (2019), 1205–1221.

[24] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. 2019. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *Proceedings of the IEEE Conference on Computer Communications*. IEEE, 2512–2520.

[25] Gang Wu, Huxing Zhang, Meikang Qiu, Zhong Ming, Jiayin Li, and Xiao Qin. 2013. A decentralized approach for mining event correlations in distributed system monitoring. *J. Parallel Distrib. Comput.* 73, 3 (2013), 330–340.

[26] Yunfan Ye, Shen Li, Fang Liu, Yonghao Tang, and Wanting Hu. 2020. EdgeFed: Optimized federated learning based on edge computing. *IEEE Access* 8 (2020), 209191–209198.

[27] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M. Alvarez, Jan Kautz, and Pavlo Molchanov. 2021. See through gradients: Image batch recovery via Gradinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16337–16346.

[28] Liekang Zeng, Xu Chen, Zhi Zhou, Lei Yang, and Junshan Zhang. 2020. CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices. *IEEE/ACM Trans. Netw.* 29, 2 (2020), 595–608.

[29] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. 2020. The secret revealer: Generative model-inversion attacks against deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 253–261.

[30] Ligeng Zhu and Song Han. 2020. Deep leakage from gradients. In *Federated Learning*. Springer, 17–31.

Received 17 October 2021; revised 14 January 2022; accepted 25 February 2022