Privacy as a Resource in Differentially Private Federated Learning

Jinliang Yuan[1], Shangguang Wang[1], Shihe Wang[1], Yuanchun Li[2*], Xiao Ma[1], Ao Zhou[1] and Mengwei Xu[1]

[1]*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, China*
[2]*Institute for AI Industry Research (AIR), Tsinghua University, China*
[*]*Shanghai AI Laboratory, China*
{yuanjinliang, sgwang, shihewang, maxiao18, aozhou, mwx}@bupt.edu.cn, liyuanchun@air.tsinghua.edu.cn

Abstract—**Differential privacy (DP) enables model training with a guaranteed bound on privacy leakage, therefore is widely adopted in federated learning (FL) to protect the model update. However, each DP-enhanced FL job accumulates privacy leakage, which necessitates a unified platform to enforce a global privacy budget for each dataset owned by users. In this work, we present a novel DP-enhanced FL platform that treats privacy as a resource and schedules multiple FL jobs across sensitive data. It first introduces a novel notion of device-time blocks for distributed data streams. Such data abstraction enables fine-grained privacy consumption composition across multiple FL jobs. Regarding the non-replenishable nature of the privacy resource (that differs it from traditional hardware resources like CPU and memory), it further employs an allocation-then-recycle scheduling algorithm. Its key idea is to first allocate an estimated upper-bound privacy budget for each arrived FL job, and then progressively recycle the unused budget as training goes on to serve further FL jobs. Extensive experiments show that our platform is able to deliver up to 2.1$\times$ as many completed jobs while reducing the violation rate by up to 55.2% under limited privacy budget constraint.**

I. Introduction

Federated learning (FL) is an emerging machine learning paradigm that enables multiple mobile devices to collaboratively learn a global model in a privacy-preserving manner. It underpins many killer applications nowadays, such as input method [1], voice assistant [2], and financial services [3]. To further protect the exchanged model update between clients and server, FL deployment is often accompanied by other privacy enhancement methods [4]–[6]. Among them, differential privacy (DP) [6] is one of the most widely adopted techniques for its lightweight overhead and flexibility. Specifically, each device introduces a local DP randomness (namely LDP) on the model update before it is sent to an untrustworthy server. The quantified randomness opens a rich trade-off between the privacy protection levels and model training accuracy.

However, a key characteristic of DP-enhanced FL is that privacy leakage can *accumulate* across multiple FL jobs [7], [8]. That is, with iterative FL jobs issued and trained on the same dataset (across clients), the private information exposed to the FL developers can increase according to the DP composition theory [7], and thus the attacking surface widens, as will be illustrated in Section II-B. Such iterative FL training is common in practice: (i) The same model could be trained through FL many times, e.g., to search for a proper model architecture [9] or regularly updating [1], [2]. (ii) Different models could be trained through FL on the same dataset. For example, the input corpora from end users could be used to fine-tune many natural language processing models such as word prediction [1], sentiment analysis [10], question answering [11], etc.

Unfortunately, we have conducted preliminary experiments and found that privacy loss in multi-job settings can cause serious performance degradation, but no prior work investigated it. They mostly consider an FL job as one shot [12]–[14], attempting to maximize the model accuracy under a privacy budget. Therefore, how to maximize the utility of sensitive data while preserving user privacy in multi-job settings is non-trivial and opens a new dimension of research opportunity.

To this end, we propose a holistic DP-based user data management platform that manages *data privacy as a resource*. On the one hand, it enables fine-grained privacy control by allowing users to specify their own privacy budget for the data they own. It also enforces the accumulated privacy leakage of each user will not exceed a pre-specified *global privacy budget*. On the other hand, it accepts FL jobs from developers and seeks to maximize the number of FL jobs successfully completed across sensitive user data. Those complete jobs are required to guarantee a service-level objective (SLO) as specified by FL developers. The platform bridges the privacy-utility gap between FL developers and users (who are also data owners). In a broader sense, we believe such a platform is a key step towards the vision of "handing over data ownership back to users" [15]–[17].

There are two unexplored challenges to enabling fine-grained privacy resource management for multiple FL jobs.

First, previous DP-based approaches todata privacy control mostly work on static, centralized datasets [18]–[20]. These approaches do not apply to our real-world FL scenario, where the data is distributed across clients and continuously expanding with new samples generated. As we will show in Section III-B, the privacy budget easily runs out with the state-of-the-art DP-enhanced FL mechanism. Therefore, the device and time dynamics in our FL scenario necessitate a new fine-grained approach todata privacy control to avoid running out of privacy budget quickly.

Second, unlike traditional hardware resources such as memory and CPU, the privacy budget is non-replenishable. Once the privacy leakage reaches the enforced budget, the dataset can never be used for training anymore. A job that is launched but not successfully completed, not only degrades the quality of service but also causes privacy budget waste. This leads to an all-or-nothing principle for the scheduling of non-replenishable privacy budget: an FL job is either allocated with enough budget to satisfy the SLO requirement (i.e., convergence accuracy), or no budget at all. However, enforcing this principle is difficult as the precise privacy budget consumption of an FL job cannot be determined in advance. This is because FL jobs are naturally integrated with client/data sampling [21], [22], of which the concrete sampling strategy cannot be obtained ahead of training with the devices dropping in/out dynamically [23]. Those unique characteristics invalidate traditional resource scheduling algorithms [24]–[26].

In this work, we propose the first unified differentially private FL platform, namely `FLScheduler`, which integrates two key techniques to address the above challenges. (i) `FLScheduler` introduces the notion of device-time blocks that splits the continuous data stream into fixed time-frame blocks on each device. Users are able to configure a desirable (ε,δ)-DP guarantee for each split block. `FLScheduler` introduces a novel time-blocks composition theorem to enforce a global (ε,δ)-DP guarantee on cross-device blocks in each given period (e.g., a day), while maintaining the on-device training on merged blocks. With this theorem, we can account for the privacy loss at the granularity of time-blocks, instead of the whole data stream. Moreover, newly generated time-blocks will be assigned a clean privacy budget, which enables endless FL jobs to be carried out as long as the data continues to come. (ii) `FLScheduler` introduces a novel, fine-grained privacy budget scheduling algorithm `AaR`, which maximizes the number of completed jobs while reducing SLO violation rate under the limited privacy budget constraint. Its key idea is to first estimate and allocate enough privacy budget (an upper bound) for each arrived job to ensure its completion with guaranteed SLO; it then progressively recycles the unused budget as more information is exposed during FL training to serve more jobs. It employs a novel weighted thriftiest job first strategy: the job with the smallest estimated privacy budget consumption is always scheduled first to enable more jobs to be completed, yet the estimated consumption is also weighted by the job's waiting time to prevent job starvation.

We have implemented `FLScheduler` atop the ShuffledFL framework [14] and performed extensive experiments to compare our `FLScheduler` with three competitive baselines. The results show that it can deliver up to 2.1× as many complete jobs while reducing the SLO violation rate by up to 55.2%.

The main contributions of this paper are as follows:

- We conduct preliminary experiments to reveal that privacy loss in multi-job settings causes serious performance degradation. Then, we propose `FLScheduler`, a novel differentially private FL platform that treats data privacy as a resource to improve the utility of sensitive user data while preserving user privacy.
- We introduce a time-blocks composition theorem to account for the privacy loss at the granularity of time-blocks, instead of the whole data stream, while enforcing a global DP guarantee. We also design `AaR`, a fine-grained privacy budget scheduling algorithm that allocates the non-replenishable privacy resource to serve more FL jobs.
- We develop a prototype of `FLScheduler` platform and conduct extensive experiments on it to demonstrate its superior performance against three baselines.

II. Background and Motivations

A. Differentially Private Federated Learning

Differential privacy is a system for sharing information about a dataset by describing the patterns of groups within the dataset while withholding information about individuals in the dataset. The common way to achieve differential privacy is introducing randomness to the computation, such that the details of individual entries can be hidden.

Definition 1. *(Differential Privacy - DP [6]). For $\varepsilon,\delta \geq 0$, a randomized mechanism $\mathscr{M} : X^n \to Y$ is said to be (ε,δ)-differentially private (in short, (ε,δ)-DP), if for all neighboring datasets $D,D' \in X^n$ and every subset $S \subseteq Y$, we have*

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

Parameters ε and δ in the above definition quantify the strength of the privacy guarantee. With a small ε, an algorithm's output gives little information about whether it ran on the original dataset or its neighboring dataset. The privacy budget ε upper-bounds the privacy loss of (ε,δ)-DP computation with probability $(1-\delta)$. Because the computation's output is much more sensitive to the assigned ε than to δ, in this paper, we will focus on ε as the sole global resource to schedule [27].

Recent studies have shown the intermediate results generated during the model training, such as updated gradients, may also expose sensitive information about the original data [28]. As for differentially private deep learning [18], the goal is to ensure that no specific data sample in the training dataset can drastically affect the model obtained by the training procedure. Such a goal is usually achieved by adding computed noise to the gradients during training. Applying DP to FL is a widely adopted approach to protect the sharing gradients from exposing the privacy of original data, which is named as differentially private FL [8]. There are two ways to employ DP in FL: the curator and local models. A trusted server is required in the curator model to collect users' raw gradients [13], and randomizes the aggregated results on the server. By contrast, users send randomized gradients (not the raw gradients) to the server in the local model approach, which does not assume a trusted party. The local randomness mechanism provides a stronger local privacy guarantee to avoid information leakage in FL scenario.

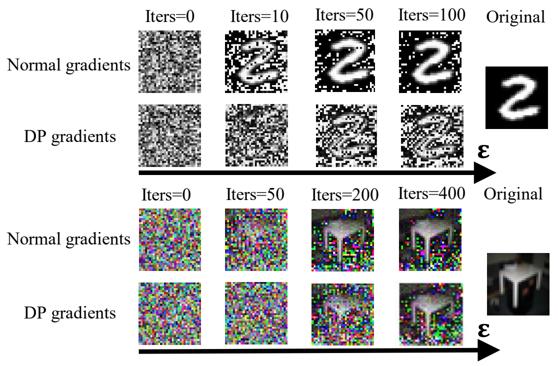


Fig. 1: Illustration of privacy leakage on MNIST [32] and CIFAR100 [33] with normal gradients and DP randomized gradients with a Gaussian noise [28] $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$. The images are reversed from the accumulated gradients using the attack method DLG [28]. The individual training samples can be easily reversed with normal gradients.

Definition 2. *(Local Differential Privacy - LDP [29]). For* $\varepsilon_0 \geq 0$*, a randomized mechanism* $\mathscr{M} : X \to Y$ *is said to be* ε_0*-local differentially private (in short,* ε_0*-LDP), if it is for every pair of input* $x, x' \in X$*, we have*

$$Pr[\mathscr{M}(x) = y] \leq e^{\varepsilon_0} Pr[\mathscr{M}(x') = y], \forall y \in Y. \qquad (2)$$

Here, ε_0 captures the privacy level, lower ε_0 means a higher privacy level. We succinctly denote the corresponding $(\varepsilon_0, 0)$-DP. However, it suffers a low utility when using ε_0-LDP mechanism to provide a stronger privacy guarantee [29].

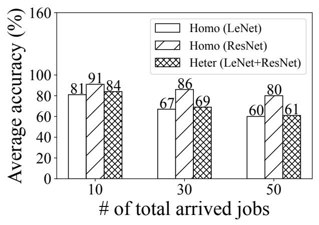
Recently, a shuffle scheme was proposed to enable significantly better privacy-utility performance [30]. The anonymization of shuffling can break the linkage from the received data at the cloud to a specific client and decouple the gradient updates sent from the same client in each iteration, thus greatly improving the privacy guarantee. The shuffle scheme has been widely applied to improve the low utility of ε_0-LDP mechanism in differentially-private FL [13], [14], [29], [31]. Among them, [14] obtained the state-of-the-art performance of higher model accuracy at the lower expense of privacy budget using a shuffled ε_0-LDP mechanism.

B. Challenges of DP in Multi-job FL

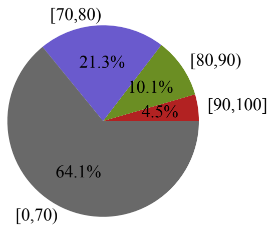
While the shuffled ε_0-LDP mechanism of [14] shows superior performance of privacy and utility trade-off, it is not able to fully exploit the utility of sensitive user data in the context of FL, due to the cumulative privacy budget introduced by frequently issued FL jobs.

Theorem 1. *(Basic DP composition [7]). For any* $\varepsilon > 0$ *and* $\delta \in [0, 1]$*, the class of* (ε, δ)*-differentially private mechanisms satisfy* $(k\varepsilon, k\delta)$*-DP under k-fold adaptive composition.*

The above composition theorem is known as a critical characteristic of DP mechanism. It shows that the composition of k queries, each of which is (ε, δ)-differentially private, is at least $(k\varepsilon, k\delta)$-differentially private. Therefore, privacy loss accumulates linearly with a sequence of DP computations, such as each round of model training with DP randomness. Figure 1 illustrates that the risk of privacy leakage is increased as the cumulative privacy budget grows.



(a) Average accuracy



(b) Accuracy distribution

Fig. 2: The accuracy of different jobs in differentially-private multi-job FL settings using the FCFS scheduling policy. (a) Average accuracy obtained with a different number of jobs. (b) Accuracy distribution with 50 heterogeneous FL jobs. Note that "homo" means the FL jobs have homogeneous models and "heter" means the models are heterogeneous.

Previous work considered FL as a one-shot process, which focuses on a single model issued once [12]–[14]. In this paper, we consider a more practical scenario that multiple FL jobs are submitted by developers to train periodically over increasing user data streams. For example, a developer may submit an image classification model that is auto-trained daily or weekly on a continuous image data stream generated as time goes by. And, other developers may leverage the same stream to train different models for object detection and recommendations.

In the following, we take a deeper look at the challenges of running multiple FL jobs through extensive experiments based on the previous work [14]. We adopt the first-come-first-serve (FCFS) algorithm to schedule dynamically arrived FL jobs. Each job is submitted to train a deep learning model (LeNet [34] or ResNet [35]) on MNIST [32].

Figure 2(a) shows that average accuracy of allocated jobs decreased significantly as the number of jobs increased. Especially as the number of arrived jobs increased to 50, the average accuracy dropped to only 61% for heterogeneous jobs. Figure 2(b) shows that only a very small number of jobs (14.6%) obtained acceptable model accuracy (e.g., $\geq$ 80%), and most jobs' accuracy (60%) is too low (e.g., $<$ 70%). Those low-precision models not only degraded the quality of service but also wasted a large portion of the privacy budget because the budget consumed by them can't be recycled for other jobs. However, prior results in [14] revealed that it can deliver higher accuracy while consuming a low budget for single job, such as 91% model accuracy of LeNet and ResNet at the expense of only $\varepsilon = 3.5$ and $\varepsilon = 2.5$. The main reason behind this serious performance degradation is that dynamically arrived jobs compete for limited privacy budget without control, resulting in an unreasonable allocation of privacy budget.

Thus, we believe that the privacy budget needs to be carefully scheduled in multi-job FL settings, in order to maximize the utility of sensitive user data while preserving privacy.

C. *System Model and Goals*

Here we describe the motivating scenario of our work in more detail. First, we assume a collection of devices that agree to participate in the FL training with a shuffled ε_0-LDP mechanism. Each device maintains a sensitive data stream, with its data continuously generated over time. User can specify a threshold of privacy budget on the stream to ensure data privacy. All operations on this stream beyond this threshold are disallowed, which provides a strong local privacy guarantee to avoid information leakage.

Second, we consider a practical scenario where multiple FL jobs are submitted by developers to train periodically over the user data streams. Each submitted job requests to train an accurate model with an expected accuracy. We specify the expected accuracy as the SLO requirement, such as a job with LeNet model must achieve an accuracy of 90%. The FL jobs dynamically arrive with different SLO requirements. We define allocated job as a job that is allocated with certain privacy budget, regardless of its expected accuracy. We define *complete job* as a job that achieved its expected accuracy. If a job is allocated some budget but does not achieve the expected accuracy, it not only degrades the quality of service but also wastes a large portion of the privacy budget, due to the non-replenishable nature of the privacy budget. We define this job as a failed job. We also define *SLO violation rate* as a ratio of failed jobs to allocated jobs, which can measure the waste portion of the privacy budget.

In this paper, our goal is to develop a unified platform that coordinates dynamically arrived FL jobs with sensitive user data streams. The platform aims to enforce a global (ε, δ)-DP guarantee across participating devices to control the leakage of user information. To improve the utility of sensitive user data, it focuses on how to schedule the global privacy budget to deliver more *completed jobs* and reduce *SLO violation rate*.

III. Differentially private FL platform

This section proposes a novel differentially private FL platform, namely `FLScheduler`, to achieve our goals. The key insight is to split the user data stream to separate data blocks with user-specified privacy budget and treat them as a resource to schedule. Moreover, we propose the time-blocks that represents the data blocks generated in a given period across all devices and enforce a global (ε, δ)-DP guarantee on them. Finally, considering the non-replenishable nature of the privacy budget, it employs a fine-grained privacy budget scheduling algorithm to schedule more FL jobs.

A. *Platform Overview*

Figure 3 shows the overview of our proposed platform. Multiple FL jobs dynamically arrive on this platform with their SLO requirements. A collection of devices within this platform introduce a user-specified $(\varepsilon_g, \delta_g)$-DP guarantee to rigorously control the access of their continuous data streams by the access controller. The server comprises two components: the scheduler schedules those submitted jobs with a limited privacy budget of the proposed time-blocks, the FL coordinator coordinates each scheduled job to run FL training using a shuffled ε_0-LDP mechanism.

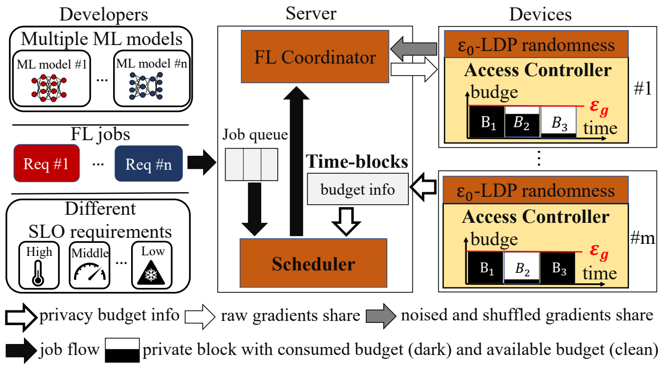


Fig. 3: Overview of our differentially private FL platform.

The access controller is introduced to manage the on-device training on user's sensitive data stream. It splits the data stream into fixed timeframe data blocks as the data generation time. Users can specify their own privacy budget for each block to ensure data privacy within this block. The introduced time-blocks represents the data blocks generated in a given period (e.g., a day) across all devices. We also design the time-blocks composition theorem to rigorously bound the cumulative leakage of data within these time-blocks when running multiple FL jobs on them. With this design, we can account for the privacy loss of cross-device FL training at the granularity of time-blocks. When the privacy loss for a time-blocks reaches the enforced $(\varepsilon_g, \delta_g)$-DP ceiling, all the blocks in this time-blocks are retired without being accessed anymore. Newly-generated data blocks are assigned a clean budget by users for continuous job training. We illustrate the time-blocks definition and time-blocks composition theorem in Section III-B and Section III-C, respectively.

The scheduler is introduced to carefully orchestrate the submitted FL jobs with the private data blocks distributed across devices. Those jobs are submitted by developers with different SLO requirements to obtain expected model accuracy. To facilitate the budget allocation, it converts the SLO expected accuracy to a reasonable parameter requirement for the LDP-based FL training process. Based on the pre-profiled behaviors, we set the requirement as (r, ε_0), which denotes the round number of FL training and LDP randomness parameter. We assume that, as long as this requirement is met, the model can be guaranteed to obtain its expected accuracy. In addition, the time-blocks composition theorem enables time-blocks level privacy accounting for multiple FL jobs while enforcing a $(\varepsilon_g, \delta_g)$-DP guarantee. However, the scheduling of this non-replenishable privacy resource differs from traditional computation resources. Therefore, we propose a novel privacy budget scheduling algorithm to fine-grained schedule these time-blocks to serve more jobs in Section IV.

B. *Global $(\varepsilon_g, \delta_g)$-DP Guarantee on Time-blocks*

`FLScheduler` seeks to maximize the utility of sensitive user data to serve more FL jobs. However, the shuffled ε_0-LDP

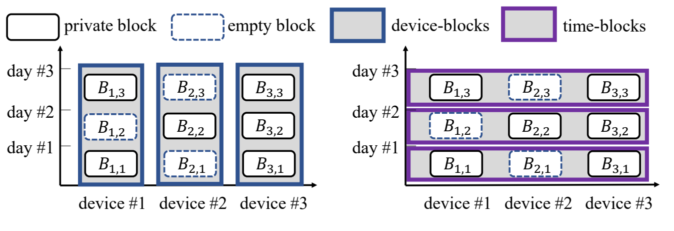


Fig. 4: Comparison of device-blocks and time-blocks on example setting: private blocks of 3 devices in 3 days.

mechanism provides a global DP guarantee across devices, which accounts for the privacy loss at the granularity of whole data streams across all devices. This leads to the critical challenge of global DP: running out of privacy budget quickly. It comprises three obstacles to our goals in specific: (1) Unused data points suffer the same privacy loss as the data points that are sampled to update the local gradients on each device. (2) Unused devices suffer the same privacy loss as the devices that are sampled to collaboratively train the global FL model. (3) Newly-generated data over time suffers the same privacy loss as the old data that has been used for training. Previous work [19] proposed a new privacy accounting method on the split data block, namely block composition theorem, to enforce a global DP guarantee on all data points in a centralized ML platform. However, the centralized block composition can not account for the privacy loss on blocks across different devices, thus cannot be applied to FL settings.

We introduce the definition of device-blocks and time-blocks that organize the private data blocks from the perspectives of device and time, respectively. We denote device-blocks as the data blocks of a device across all timeframes, and time-blocks as the data blocks generated in a given period across all devices. We illustrate the two types of data blocks with an example in Figure 4. The data generated on each device is split into timeframe data blocks for a given period, like one day a block. A device-blocks denotes a collection of data blocks generated on a device over continuous periods, which is naturally maintained on this device. A time-blocks denotes a collection of data blocks generated on the same day, but on different devices as shown in Figure 4(b). We propose a new time-blocks composition theorem to enforce a global DP guarantee on each time-blocks in the next subsection. The global privacy budget of each time-blocks must not exceed the minimum user-specified budget of each block with this time-blocks. Both the structure and budget information of time-blocks are maintained on server for the subsequent scheduling.

C. Time-blocks Composition Theorem

This section presents the time-blocks composition theorem. It is introduced to provide privacy accounting at the granularity of time-blocks when running multiple FL jobs with different shuffled ε_0-LDP mechanisms. The new privacy accounting method is used to address the challenge of running out of privacy budget quickly by iterative FL jobs.

The classical DP composition theorem, as shown in Theorem 1, proves that a (ε, δ)-DP mechanism iterates r rounds on the same dataset is $(\sum_{i=1}^{r} \varepsilon_i, \sum_{i=1}^{r} \delta_i)$-DP, where $(\varepsilon_i, \delta_i)$ is the DP parameter at round i. `FLScheduler` adopts a shuffled ε_0-LDP mechanism [14] that yields significant improvement of approximate (ε, δ)-DP guarantee. Below, we provide a formal statement of that result from Theorem 3 of previous work [14].

Lemma 1. *For a shuffled ε_0-LDP mechanism $\mathscr{M}$, if we run $\mathscr{M}$ over r iterations, then we have $\mathscr{M}$ is (ε, δ)-DP, where $\delta > 0$ and ε is upper-bounded by:*

$$\varepsilon \leq f(r, \varepsilon_0). \tag{3}$$

The above result provides a privacy accounting at the granularity of whole data streams in `FLScheduler`, which is related to the round number of FL training r and LDP randomness parameter ε_0. Then, we extend the block composition theorem [19] to prove that the privacy loss over the entire dataset combined with multiple sampled time-blocks is the same as the maximum privacy loss on each sampled time-blocks. This enables the local update on the device-blocks combined with sampled on-device blocks, but still accounts for the privacy loss on the sampled time-blocks. Applying Lemma 1 and extended result, we give the upper bound of consumed privacy budget for single job when running shuffled ε_0-LDP mechanism on sampled time-blocks.

Theorem 2. *(Upper bound of consumed privacy budget for single job) For a shuffled ε_0-LDP mechanism $\mathscr{M}$, if a job run r rounds of FL training using $\mathscr{M}$ with multiple sampled time-blocks at each round, then the consumed privacy budget on any sampled time-blocks t is upper-bounded by:*

$$\varepsilon^t \leq f(r', \varepsilon_0), \text{ and } r' \leq r, \tag{4}$$

where r' represents the actual rounds that this job sampled time-blocks t during total r rounds. For example, if this job sampled this time-blocks every time, then we have $r' = r$. The implication of Theorem 2 is that privacy loss is accounted at the granularity of individual time-blocks that is actually used. And the precise consumed budget is related to the round number of FL training that is actually sampled. In addition, the un-sampled time-blocks are not accounted for the privacy loss, which means that newly-generated data blocks can be assigned a clean budget to allow continuous training.

Next, we introduce the composition theorem for multiple FL jobs on time-blocks.

Theorem 3. *(Time-blocks composition for multiple jobs) For different shuffled ε_0^k-LDP mechanisms, if n submitted jobs with requirements $\{(r_k, \varepsilon_0^k)\}_{k=1}^{n}$ and each job k requests to run r_k rounds of FL training with LDP parameter ε_0^k on multiple sampled time-blocks at each round, then for any time-blocks at any round r, the access controller enforces the total consumed privacy budget on this time-blocks:*

$$\sum_{k=1}^{n} f(r_k', \varepsilon_0^k) \leq \varepsilon_g, \text{ and } r_k' \leq r, \tag{5}$$

where r'_k represents the actual rounds that job k samples this time-blocks during total r rounds, and ε_g denotes the enforced global privacy budget of this time-blocks that pre-specified by users when its blocks are generated. The implication of Theorem 3 is that we can still account for the privacy loss at the granularity of time-blocks, even with dynamically arrived FL jobs. This enables fine-grained privacy budget scheduling of continuously generated time-blocks for more submitted FL jobs as illustrated in next section.

IV. Privacy Budget Scheduling

FLScheduler incorporates the time-blocks as a new resource unit to coordinate multiple FL jobs while enforcing a global (ε, δ)-DP guarantee. This section explores how to schedule such resource to maximize the number of complete jobs while reducing the SLO violation rate. We first describe why existing scheduling algorithms are not adequate in our scenario. Then, we present the design of our new algorithm, namely AaR (**A**llocate **a**nd **R**ecycle).

A. Design Considerations

As discussed in Section II-B, privacy is a scarce resource that should be carefully scheduled. However, the scheduling scenario of our FLScheduler substantially differs from traditional resource scheduling systems in the following aspects.

- First, unlike traditional resources such as CPU time cycles and memory footprint, privacy is a non-replenishable resource. Once the consumed budget reaches the enforced global privacy budget, the corresponding time-blocks can never be used for training anymore.
- Second, an efficient privacy scheduler shall obey an *all-or-nothing* principle, i.e., each FL job is either allocated with enough privacy budget to satisfy the SLO (expected accuracy), or no budget at all. A scheduled job that violates this principle, not only degrades the quality of service but also wastes the privacy budget.
- Third, FL is naturally integrated with client/data block sampling. However, the concrete sampling strategy is often determined on demand (i,e, ahead of each global round) because the device can drop in/out from time to time. Without apriori knowledge of which data will be used for training, the scheduling algorithm design becomes more complicated.

The above design considerations are ignored by the scheduling algorithms on traditional hardware resources. For example, existing resource schedulers [24], [25], [36], [37] in datacenters deal with replenishable resources such as CPU and memory. A few recent scheduling algorithms [36], [37] specifically designed for non-replenishable resource do not consider the all-or-nothing principle. A recently proposed PrivateKube [27] considers the scheduling of non-replenishable privacy resource for centralized ML workloads. The proposed dominant private block fairness algorithm (DPF) scheduled the privacy budget of private blocks to improve ML system throughput. However, it can not fully exploit the utility of privacy budget in our scenario as it presumes a fixed selection of private blocks.

B. Algorithm Design

We propose AaR, the first-of-its-kind scheduling algorithm for privacy resources in multi-job FL scenarios. It consists of two core stages: a pre-allocation of an estimated upper-bound privacy budget at each FL job arrival; and a progressive recycling of the un-consumed privacy budget during FL training. The rationales are twofold. First, the block sampling strategy makes it impractical to obtain precise privacy budget requirements for each time-blocks ahead of its execution. With the aid of estimated upper bound in Theorem 2, the pre-allocation can provide enough budget to guarantee the all-or-nothing scheduling. Such a design helps to reduce the SLO violation rate. Second, the upper bound of pre-allocation may lead to a large portion of privacy budget waste, due to the actual block sampling during the training process. With the progressive recycling of the un-consumed budget, the excessively allocated privacy can be recycled as the training goes on, therefore more FL jobs can be possibly scheduled.

We next present the detailed workflow of AaR, with its pseudocode shown in Algorithm 1. For each time-blocks, we define five budget fields: (1) ε^G denotes the initial budget as its first creation, where $\varepsilon^G = \varepsilon_g$. (2) ε^U, called unlocked budget, which denotes the unlocked budget as the jobs arrive. (3) ε^L, called locked budget, where $\varepsilon^U + \varepsilon^L = \varepsilon^G$. (4) ε^{AL}, called allocated budget, is the budget that has been allocated to jobs, where $\varepsilon^{AL} \leq \varepsilon^U$. (5) ε^{AV}, called available budget, is the available budget that has been unlocked, where $\varepsilon^{AL} + \varepsilon^{AV} = \varepsilon^U$. For each submitted job, we define two demand vectors: *total demand budget* and *next-round demand budget*, denoted as d_T and d_N. The former represents a demand vector of privacy budget on all time-blocks of this job to complete the FL training as required (r, ε_0). The latter represents a demand vector of privacy budget on all time-blocks of this job to run the next round of FL training using a shuffled ε_0-LDP mechanism.

AaR gradually unlocks the initial privacy budget as jobs arrive (function OnArriveJob, Line 2): a new job with the requirement $(r,\ \varepsilon_0)$ to obtain SLO expected accuracy. We define a fair share privacy budget over the first arrived n jobs: $\varepsilon^{FS} = \varepsilon^G / n$. As a job in the first n jobs arrives, we unlock a fair share budget ε^{FS} on all time-blocks. The unlocked budget is also used to update the available budget ε^{AV} and unlocked budget ε^U (Line 18). Each arrived job is first scheduled into a waiting queue (Line 3). Gradually unlocking the privacy budget is key to dealing with a non-replenishable resource in a dynamic setting.

When the waiting queue adds a new job, the scheduler adopts a weighted-thriftiest-job-first strategy to schedule those dynamically arrived jobs. This strategy considers the waiting time and *total demand budget* to sort the jobs in the waiting queue, which is defined as: $\beta \cdot \frac{d_T}{waiting\_time}$ (Line 4). Then, the job with the smallest d_T tends to be scheduled first, yet it is also weighted by the job's waiting time to schedule the job with a larger *waiting_time*. We introduce a parameter β to balance the two aspects of allowing more jobs and avoiding job starvation. Moreover, it is impractical to obtain the precise

Algorithm 1: Allocate and Recycle (AaR)

```
input : the first $n$ jobs, $waiting\_queue$, $running\_queue$,
        $\varepsilon^{AV} = \varepsilon^{AL} = \varepsilon^{U} = 0$, $\varepsilon^{L} = \varepsilon^{G} = \varepsilon_g$
1  Function OnScheduleJob():
2      OnArriveJob(J)
3      schedule $J$ to waiting_queue
4      SortBy(waiting_queue, $d_T$, waiting_time)
       // strict scheduling stage
5      for each J in sorted waiting_queue do
6          if $d_T \leq \varepsilon^{AV}$ then
7              Allocate($d_T$), and schedule $J$ to running_queue
8      OnRunJob(running_queue)
       // best-effort scheduling stage
9      for each J in sorted waiting_queue do
10         for each round $i \leftarrow 0$ to $r$ do
11             if $d_N \leq \varepsilon^{AV}$ then
12                 Allocate($d_N$)
13                 $T, D \leftarrow$ SampleBlocks(J)
14                 Local update on $D$, privacy loss on $T$
15                 Recycle($T$)
16             Else stop the scheduling
17 Function OnArriveJob(Job J):
18     $\varepsilon^{U} = \varepsilon^{U} + \frac{\varepsilon^{G}}{n}$, $\varepsilon^{AV} = \varepsilon^{AV} + \varepsilon^{U}$
19 Function Allocate($d_T$):
20     $\varepsilon^{AL} = \varepsilon^{AL} + d_T$, $\varepsilon^{AV} = \varepsilon^{AV} - \varepsilon^{AL}$
21 Function OnRunJob(running_queue):
22     for each J in running_queue in parallel do
23         for each round $i \leftarrow 0$ to $r$ do
24             $T, D \leftarrow$ SampleBlocks(J)
25             Local update on $D$, privacy loss on $T$
26             Recycle($T$, $d_N$)
27 Function SampleBlocks(Job J):
28     $T \leftarrow$ sample time-blocks
29     $D \leftarrow$ sample on-device blocks in each selected $T$
30 Function Recycle($T$):
31     for each time-blocks $t \notin T$ do
32         $\varepsilon^{t} \leftarrow$ pre-allocated but un-consumed privacy budget
33         $\varepsilon^{AL} = \varepsilon^{AL} - \varepsilon^{t}$, $\varepsilon^{AV} = \varepsilon^{AV} + \varepsilon^{t}$
```

d_T for the uncertain time-blocks sampling. So, we follow Theorem 2 to estimate the upper bound of demand privacy budget on each time-blocks in demand vector d_T. Then, it compares each job's d_T in the waiting queue to the current available budget ε^{AV} in order (Lines 5-7). A job with the smallest d_T is scheduled first, and then the next one. The scheduled job will be allocated with all of the demanded privacy budget on all time-blocks at once. The allocated budget is also used to update the available budget ε^{AV} and allocated budget ε^{AL} (function Allocate, Line 20). After that, this job will be scheduled to the running queue.

When the running queue receives a new job, the FL coordinator coordinates this job to run FL training with shuffled ε_0-LDP Mechanism (function OnRunJob, Line 8,21-26). Note that this coordination is executed independently and parallelly among these jobs in the running queue, which results from the pre-allocation guided by the estimated upper bound strategy. Then, for each job, it employs a block sampling strategy to improve the performance of privacy and utility trade-off (function SampleBlocks, Line 24). It first samples partial time-blocks, and then samples partial on-device data blocks from each sampled time-blocks. After the sampling, it distributes the model to local update on those sampled device-blocks D, and accounts for the privacy loss on those sampled time-blocks T (Line 25). Then, it recycles the pre-allocated but un-consumed privacy budget of un-sampled time-blocks ($\notin T$) (function Recycle, Line 26). Due to the control of the access controller in Section III-A, this job doesn't consume the privacy budget of these un-sampled time-blocks. Thus, the pre-allocated but un-consumed privacy budget are recycled to update the available budget ε^{AV}, and used to re-schedule more jobs (Lines 31-33).

When all n jobs arrive and all running jobs finish, if the available budget still does not satisfy the smallest *total demand budget*, we relax the all-or-nothing guarantee to schedule the jobs in the waiting queue on a best-effort basis. (Lines 9-15). A job will be scheduled to run only a round, if the available budget ε^{AV} is larger than its *next-round demand budget* d_N (Line 11). The budget allocated at one time is just enough to run the next round of training (Line 12). Other processes are the same as mentioned before (Lines 13-15).

AaR halts when there is not enough available budget to run a round for the waiting job with the smallest d_T (Line 16).

V. Evaluation

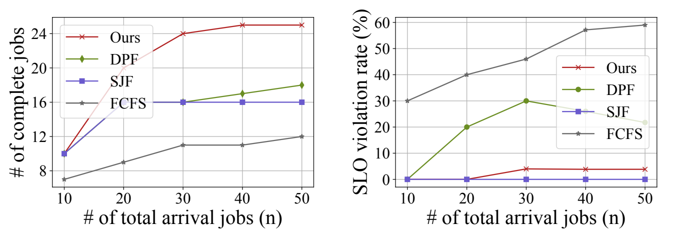
A. Experiment Settings

We have built a prototype of FLScheduler atop ShuffledFL [14], which consists of 4 major functional modules: a LDP-based FL training with shuffled ε_0-LDP mechanism; a data preprocessing module that organizes training data sets as the form of device-blocks and time-blocks; a privacy accounting module to account for the privacy loss on time-blocks during the training process; a scheduler module that adopts various algorithms to schedule the dynamic arrival jobs with designed time-blocks.

We compare FLScheduler to three baseline algorithms.

- First-come-first-serve (FCFS) that always allocates resources for jobs by their order of arrival to the platform.
- Short-job-fisrt (SJF) that schedules jobs by their demands to achieve the expected accuracy while considering all-or-nothing principle, but without private blocks design.
- dominant-private-block-fairness (DPF) [27] that treats each private block as a separate resource and allocates requested blocks all-or-nothing to ensure their accuracy goals. The main difference of DPF, compared to AaR, is a prior and fixed time-blocks selection, which is a coarse-grained algorithm without considering the unique characteristics of FL scenario.

We mainly report two evaluation metrics: number of *completed jobs* and *SLO violation rate*, which are explained in detail in Section II-C. We assume a random arrival of submitted FL jobs at each time slot. In this experiment, we use the FL training round as the time slot and each job arrives to be scheduled at a round in total 4,000 rounds. We generate two types of job that trains ML model LeNet [34] and ResNet [35]. We split the total arrival jobs as 25% to 75% with respective requirements (r, ε_0) of $(2{,}000, 1.5)$ and $(1{,}000, 1.5)$,



(a) Increase complete jobs (b) Reduce violation rate

Fig. 5: End-to-end performance of `FLScheduler` ("Ours") compared to baselines under a different number of arrived jobs.

Algos	V_Rate	Number of jobs		Average accuracy	
		Complete	Allocated	Complete	Allocated
Ours	3.8%	25	26	87.6%	84.6%
DPF	21.7%	18	23	86.4%	85.6%
SJF	0	16	16	**90.5%**	**90.5%**
FCFS	**59%**	**12**	30	85.7%	71%

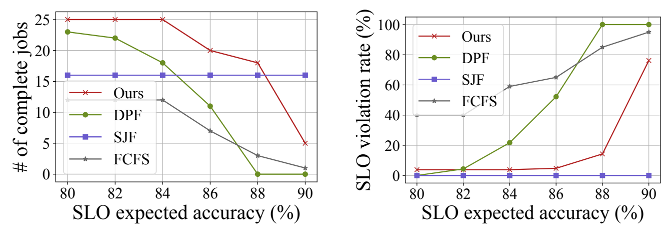
TABLE I: End-to-end results in Figure 5 when $n = 50$. Note that the V_Rate denotes the SLO violation rate.

by default. The estimated upper bound of privacy budget is $\varepsilon = 3.5$ and $\varepsilon = 2.5$, respectively. The initialized global privacy budget is $\varepsilon_g = 40$ throughout this experiment. We extend and split the MNIST [32] dataset to deploy 10,000 devices with 50 samples on each device. The 50 samples are split into 10 time-blocks with each having 5 samples. We randomly sample a various number of time-blocks and 100 data blocks within each sampled time-blocks for each round of FL training.

B. End-to-end Results

In this section, we show the end-to-end performance of `FLScheduler` and three baselines under a different number of total arrived jobs (10-50) and SLO expected accuracy (80%-90%). Figure 5 and Table I show the results under different arrived jobs with the expected accuracy 84%. We make following key observations. `FLScheduler` significantly outperforms FCFS's: 2.1× as many complete jobs and 55.2% lower SLO violation rate from Table I. Because the early arrived jobs selfishly consume most of the budget, and thus leave not-enough budget for subsequent jobs to achieve their expected accuracy. Compared to SJF, `FLScheduler` achieves many more completed jobs and a proximate violation rate from Figure 5. As shown in Table I, `FLScheduler` is able to grant up to 56.3% more completed jobs than SJF, but with only 2.8% average accuracy degradation. `FLScheduler`'s violation rate is only 3.8%. Yet, the violation rate of SJF is 0, because it schedules FL jobs with all samples, therefore delivering higher accuracy at the expense of too much privacy budget, and fewer completed jobs.

Compared to DPF, `FLScheduler` achieves a much lower violation rate with a significant improvement of complete jobs from Figure 5. Specifically, in Table I, `FLScheduler` reduces the violation rate by 17.9% and increases the complete jobs by 38.9%, compared to DPF. The average accuracy of allocated jobs in `FLScheduler` is only 1% lower than that



(a) Number of complete jobs (b) SLO violation rate

Fig. 6: End-to-end performance of `FLScheduler` ("Ours") compared to baselines under different SLO expected accuracy.

in DPF, even though DPF selects 20% more time-blocks. Such performance degradation is mainly attributed to the fixed time-blocks selection mechanism in DPF. This mechanism causes jobs in DPF to either decrease model accuracy due to fewer selected time-blocks, or more selected time-blocks to decrease the number of complete jobs. In contrast, `FLScheduler` supports device-time block sampling strategies, which enables better performance of privacy-utility trade-off in Section IV-B.

Figure 6 shows `FLScheduler`'s end-to-end performance, compared to three baselines under different levels of expected accuracy when $n = 50$. As the expected accuracy increases, the number of complete jobs decreases and the violation rate increases for each algorithm. However, `FLScheduler` still delivers better performance than baselines. As shown in Figure 6(a), even with a higher expected accuracy 88%, `FLScheduler` allows 18 completed jobs, but the values of FCFS and DPF are as low as 4 and 0. Figure 6(b) shows that `FLScheduler`'s violation rate is only 14.2% at the expected accuracy 88%, which is as much as 70.8% and 85.8% lower than FCFS and DPF. All the jobs scheduled by SJF can achieve their expected accuracy, but the number of complete jobs is only 16, and unchanged even with lower expected accuracy. The reason is that SJF does not adopt private blocks to enabling fine-grained privacy budget scheduling.

C. Impact of Time-blocks Selection Rate

The time-blocks sampling complicates the design of our scheduling algorithm to achieve more complete jobs with a lower violation rate. Therefore, we further conduct experiments with different values of time-blocks selection rate.

Figure 7 shows that `FLScheduler` achieves better trade-off between the performance among completed jobs and violation rate. Specifically, `FLScheduler` achieves more or equal completed jobs than DPF under all settings. Moreover, with fewer selected time-blocks such as 20% and 40%, `FLScheduler` even delivers 4× and 29× more complete jobs than DPF. For the violation rate, `FLScheduler` reduces by 84%, 76% and 18% with fewer selected time-blocks, and only a little higher when time-blocks selection rate $\geq$ 80%. The rationale is that, compared to the prior and fixed blocks sampling of DPF, `FLScheduler`'s device-time blocks sampling strategy allows the model to be trained by more samples at a lower expense of privacy budget.

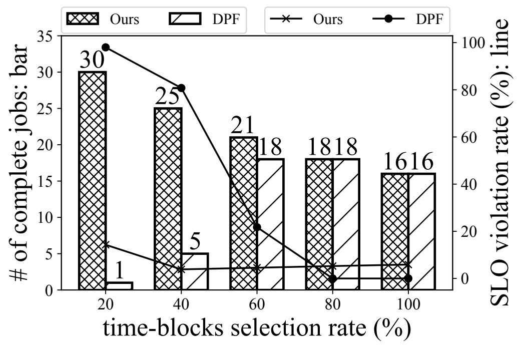


Fig. 7: Impact of time-blocks selection rate on `FLScheduler` ("Ours") and DPF when *n*=50 and SLO expected accuracy is 84%. Bar: higher is better; line: lower is better.

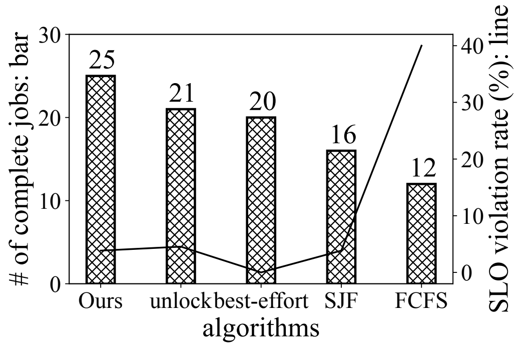


Fig. 8: `FLScheduler`'s ablation study on two key techniques: gradually unlock budget and best-effort scheduling. Unlock: `AaR` (w/o unlock); best-effort: `AaR` (w/o best-effort).

D. Ablation Results

We also break down `FLScheduler`'s components to perform an ablation study. The results highlight the effectiveness of its two key incorporated components. (1) `AaR` w/o unlock. We disable the design of gradually unlocking privacy budget as jobs arrive. As such, more budget will be allocated to jobs that arrive first, but not be allocated by a fair share budget. (2) `AaR` w/o best-effort. We remove the best-effort scheduling, which employs next-round demand budget to schedule only one round for one job at a time.

The results in Figure 8 show their compared performance to `FLScheduler` ("Ours"), SJF and FCFS algorithms. For the number of complete jobs, `AaR` w/o best-effort and `AaR` w/o unlock still outperform SJF and FCFS, but are 20% and 16% less than ours with all techniques enabled. For the violation rate, `AaR` w/o unlock achieves similar performance as ours, while `AaR` w/o best-effort slightly outperforms ours.

VI. Related Work

Recent studies have shown that FL's approach of sending only model parameters can also leak the privacy of the raw data [28], [38]–[41]. To decrease the risk of leaking participants' sensitive information, multiparty computing [4], homomorphic encryption [42], and differential privacy [6], [18] are used to protect the unsafe gradients sharing. Among them, DP [6] is one of the most widely adopted techniques for its lightweight overhead and flexibility. FL with a central DP mechanism [12], [43] assumes a trusted server to randomize the aggregated model, which provides higher model accuracy but weaker privacy guarantee. Other FL works adopt a local DP mechanism to randomize the on-device updated gradients locally, which provide a stronger local privacy guarantee but lower model accuracy [8], [13], [29]. Recently, a shuffle scheme was proposed to enable significantly better privacy-utility performance [30]. Several classic previous works introduce the anonymization of shuffle scheme to improve the low utility of FL with local DP mechanism [14], [29], [31].

However, the above works considered FL as a one-shot process. In this paper, we take a more practical assumption that multiple FL jobs are submitted by developers to train periodically over increasing user data streams. Decades of existing works focus on scheduling computation resources, such as CPU and memory, but barely no work studies the scheduling of DP privacy budget. Previous FL works [44]–[46] mainly focus on energy-aware or limited computation or storage, with assuming to decouple the privacy protection and efficiency improvement. These work are not suitable for our differentially private FL platform. There are two similar works considering non-replenishable resources scheduling [36], [37]. But, neither of them can achieve the required all-or-nothing scheduling of the privacy resource. Furthermore, they assume static arrived jobs, which is not suitable for the dynamic setting in our platform. A novel algorithm proposed recently, dominant private block fairness (DPF) [27], is the closest work to ours. The main idea is to schedule privacy budget for ML workloads with the introduced private block abstraction [19]. However, it made an impractical assumption that a prior and fixed selection of blocks can achieve optimal model utility. This only provides a coarse-grained privacy budget scheduling, which may lead to a serious waste of privacy resource.

VII. Conclusion

`FLScheduler` is the first unified differentially private FL platform that treats and manages data privacy as a resource. It incorporates two key techniques: a time-blocks composition theorem that enables privacy accounting at the granularity of time-blocks, instead of the whole data stream. a fine-grained privacy budget scheduling algorithm that maximizes the number of completed jobs while reducing SLO violation rate under the limited privacy budget constraint. Experiments show that `FLScheduler` can deliver up to 2.1× as many complete jobs while reducing the SLO violation rate by 55.2%.

VIII. Acknowledges

This work was supported by National Key R&D Program of China (No.2021ZD0113001), NSFC (62102045, U21B2016, 61922017, 62032003, 61921003, 61902036 and 62272261), Beijing Nova Program (No.Z211100002121118), and Young Elite Scientists Sponsorship Program by CAST (No.2021QNRC001).

References

[1] Mengwei Xu, Feng Qian, Qiaozhu Mei, Kang Huang, and Xuanzhe Liu. Deeptype: On-device deep learning for input personalization service with minimal privacy concern. In *Proceedings of Interactive, Mobile, Wearable and Ubiquitous Technologies*, pages 197:1–197:26, 2018.

[2] Filip Granqvist, Matt Seigel, Rogier C. van Dalen, Áine Cahill, Stephen Shum, and Matthias Paulik. Improving on-device speaker verification using federated learning with privacy. In *Proceedings of International Speech Communication Association*, pages 4328–4332, 2020.

[3] Wensi Yang, Yuhang Zhang, Li Li, and Cheng-Zhong Xu. Ffd: A federated learning based method for credit card fraud detection. In *Proceedings of International conference on big data*, pages 18–32, 2019.

[4] Manas A. Pathak, Shantanu Rane, and Bhiksha Raj. Multiparty differential privacy via aggregation of locally trained classifiers. In *Proceedings of Conference on Neural Information Processing Systems*, 2010.

[5] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *Trans. Inf. Forensics Secur.*, 13(5):1333–1345, 2018.

[6] Cynthia Dwork. Differential privacy. In *Encyclopedia of Cryptography and Security, 2nd Ed.* 2011.

[7] Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. Boosting and differential privacy. In *Proceedings of Foundations of Computer Science*, pages 51–60, 2010.

[8] Ruixuan Liu, Yang Cao, and Masatoshi Yoshikawa. FLAME: differentially private federated learning in the shuffle model. In *Proceedings of Conference on Artificial Intelligence*, pages 8688–8696, 2021.

[9] Chaoyang He, Murali Annavaram, and Salman Avestimehr. Fednas: Federated deep learning via neural architecture search. *arXiv preprint arXiv:2004.08546*, 2020.

[10] Han Qin, Guimin Chen, Yuanhe Tian, and Yan Song. Improving federated learning for aspect-based sentiment analysis via topic memories. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, pages 3942–3954, 2021.

[11] Betty van Aken, Benjamin Winter, Alexander Löser, and Felix A. Gers. How does BERT answer questions?: A layer-wise analysis of transformer representations. In *Proceedings of International Conference on Information and Knowledge Management*, pages 1823–1832, 2019.

[12] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *Proceedings of International Conference on Learning Representations*, 2018.

[13] Lichao Sun and Xun Chen. LDP-FL: practical private aggregation in federated learning with local differential privacy. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2021.

[14] Antonious Girgis and Suhas Diggavi. Renyi differential privacy of the subsampled shuffle model in distributed learning. In *Proceedings of Conference on Neural Information Processing Systems*, 2021.

[15] California consumer privacy act. https://oag.ca.gov/privacy/ccpa, 2018.

[16] General data protection regulation. https://gdpr-info.eu, 2019.

[17] Handing ownership of data back to the users. https://analyticsindiamag.com/web-3-0-handing-ownership-of-data-back-to-the-users, 2021.

[18] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of Conference on Computer and Communications Security*, 2016.

[19] Mathias Lécuyer, Riley Spahn, Kiran Vodrahalli, Roxana Geambasu, and Daniel Hsu. Privacy accounting and quality control in the sage differentially private ML platform. In *Proceedings of Symposium on Operating Systems Principles*, 2019.

[20] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. A first look at deep learning apps on smartphones. In *The Web Conference*, 2019.

[21] Chenning Li, Xiao Zeng, Mi Zhang, and Zhichao Cao. Pyramidfl: A fine-grained client selection framework for efficient federated learning. In *Proceedings of International Conference on Mobile Computing and Networking*, 2022.

[22] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *Proceedings of Operating Systems Design and Implementation*, pages 19–35, 2021.

[23] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, pages 50–60, 2020.

[24] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. Altruistic scheduling in multi-resource clusters. In *Proceedings of Operating Systems Design and Implementation*, 2016.

[25] David Shue, Michael J. Freedman, and Anees Shaikh. Performance isolation and fairness for multi-tenant cloud storage. In *Proceedings of Operating Systems Design and Implementation*, 2012.

[26] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of Symposium on Networked Systems Design and Implementation*, 2011.

[27] Tao Luo, Mingen Pan, Pierre Tholoniat, Asaf Cidon, Roxana Geambasu, and Mathias Lécuyer. Privacy budget scheduling. In *Proceedings of Operating Systems Design and Implementation*, 2021.

[28] Ligeng Zhu and Song Han. Deep leakage from gradients. In *Proceedings of Conference on Neural Information Processing Systems*, 2019.

[29] Antonious M. Girgis, Deepesh Data, Suhas N. Diggavi, Peter Kairouz, and Ananda Theertha Suresh. Shuffled model of differential privacy in federated learning. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2021.

[30] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. Private summation in the multi-message shuffle model. In *Proceedings of Conference on Computer and Communications Security*, 2020.

[31] Vitaly Feldman, Audra McMillan, and Kunal Talwar. Hiding among the clones: A simple and nearly optimal analysis of privacy amplification by shuffling. In *Proceedings of Symposium on Foundations of Computer Science*, 2022.

[32] Mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[33] Cifar-100 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[34] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[36] Ian A. Kash, Ariel D. Procaccia, and Nisarg Shah. No agent left behind: Dynamic fair division of multiple resources. *Journal of Artificial Intelligence Research*, 51:579–603, 2014.

[37] David C. Parkes, Ariel D. Procaccia, and Nisarg Shah. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. *Transactions on Economics and Computation*, 3:1–22, 2015.

[38] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In *Proceedings of Conference on Computer and Communications Security*, 2017.

[39] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *Proceedings of Symposium on Security and Privacy*, 2019.

[40] Chengxu Yang, Qipeng Wang, Mengwei Xu, Zhenpeng Chen, Kaigui Bian, Yunxin Liu, and Xuanzhe Liu. Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. In *The Web Conference*, 2021.

[41] Dongqi Cai, Yaozong Wu, Shangguang Wang, Felix Xiaozhu Lin, and Mengwei Xu. Autofednlp: An efficient fednlp framework. *arXiv preprint arXiv:2205.10162*, 2022.

[42] Chengliang Zhang, Suyi Li, Junzhe Xia, Feng Yan, and Yang Liu. Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In *Proceedings of Annual Technical Conference*, 2020.

[43] Lu Miao, Wei Yang, Rong Hu, Lu Li, and Liusheng Huang. Against backdoor attacks in federated learning with differential privacy. In *International Conference on Acoustics, Speech and Signal Processing*, pages 2999–3003, 2022.

[44] Wei Yang Bryan Lim, Jer Shyuan Ng, Zehui Xiong, Chunyan Miao, and Dong In Kim. Dynamic edge association and resource allocation in self-organizing hierarchical federated learning networks. *Journal on Selected Areas in Communications*, 39:3640–3653, 2021.

[45] Wei Yang Bryan Lim, Jer Shyuan Ng, Zehui Xiong, Jiangming Jin, Yang Zhang, Dusit Niyato, Cyril Leung, and Chunyan Miao. Decentralized edge intelligence: A dynamic resource allocation framework for hierarchical federated learning. *Transactions on Parallel and Distributed Systems*, 33:536–550, 2022.

[46] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. Deepcache: Principled cache for mobile deep vision. In *Proceedings of International Conference on Mobile Computing and Networking*, pages 129–144, 2018.