Noname manuscript No.
(will be inserted by the editor)

SmartDL: Energy-Aware Decremental Learning in a Mobile-based Federation for Geo-spatial System

Wenting Zou · Li Li · Zichen Xu* · Dan Wu* · ChengZhong Xu · Yuhao Wang · Haoyang Zhu · Xiao Sun

Received: date / Accepted: date

Abstract Federated learning is designed to collaboratively train a shared model based on a great many mobile devices while protecting data privacy, which has been widely adopted to support different geo-spatial systems. However, two critical issues prevent federated learning to be effectively deployed on resource-constrained devices in large scale. First, federated learning causes high energy consumption which can badly hurt the battery lifetime of mobile devices. Second, leakage of sensitive personal information still occurs during the training process. Thus, a system that can effectively protect the sensitive information while improving the energy efficiency is urgently required for a mobile-based federated learning system.

The corresponding authors are Zichen Xu (xuz@ncu.edu.cn) and Dan Wu (wudan@ncu.edu.cn). The first two authors contribute equally to this work. Wenting Zou was a visiting student in SIAT.
Wenting Zou
Nanchang University, China CITIC Bank

Li Li
ShenZhen Institute of Advanced Technology, Chinese Academy of Sciences

Zichen Xu
Nanchang University

Dan Wu
Nanchang University

ChengZhong Xu
University of Macau

Yuhao Wang
Nanchang University

Haoyang Zhu
Institute of Systems Engineering

Xiao Sun
Institute of Systems Engineering

This paper proposes SmartDL, an energy-aware decremental learning framework that well balances the energy efficiency and data privacy in an efficient manner. SmartDL improves the energy efficiency from two levels: 1) global layer, which adopts an optimization approach to select a subset of participating devices with sufficient capacity and maximum reward. 2) local layer, which adopts a novel decremental learning algorithm to actively provides the decremental and incremental updates, and can adaptively tune the local DVFS at the same time. We prototyped SmartDL on physical testbed and evaluated its performance using several learning benchmarks with real-world traces. The evaluation results show that compared with the original federated learning, SmartDL can reduce energy consumption by 75.6%-82.4% in different datasets. Moreover, SmartDL achieves a speedup of 2-4 orders of magnitude in model convergence while ensuring the accuracy of the model.

Keywords Federated Learning · Data Privacy · Energy Management · Mobile Computing

1 Introduction

Federated learning (FL) [1] is a booming distributed learning technique which collaboratively trains a deep learning model based on the data located on multiple devices. During the overall training process, the private data is always saved on the local devices to protect the privacy. Thus, the FL technique has been widely adopted to support different kinds of privacy-concerned applications, such as Geospatial Information Analysis [2] etc. Specifically, the common feature of these applications is that the participating devices only communicate with the central server using the meta data, e.g., model gradients. In this way, federated learning can alleviate the problem of privacy leak to a certain extent. Internet companies such as Google, Amazon, and Facebook, use the federated learning framework in their learning to conquer the privacy conscious data market with a net worth of $200 billion [3].

Under the premise of protecting user privacy, federated learning also supports the co-building models between multiple devices. In a federated learning based geo-spatial system, each device can save a copy of local geo-spatial data object and different versioned model. Usually, a small group of devices are called *servers*, which can publish the master model and calculate the alignment between all trained local models. The remaining devices are called *workers*, which subscribe models from *servers* and repeat the training process in-synchronous manner (i.e., *federation*). Therefore, federated learning can converge after retraining the target model in a sufficient time without exchanging original data objects among devices. In theory, the federated learning framework is expected to expand to millions to billions of devices [4].

In fact, due to the following two main reasons, this simple federated learning framework can be expensive and may still leak privacy. First, the original federated learning implicitly assumes that the model to be trained is simple enough and all synchronized *workers* can make a training response within a specified time, so that the *servers* can complete the model convergence computation within a given time threshold. However, this kind of collaborative learning behavior can reveal the private geo-spatial information of users through model inverse reasoning. In addition, the model itself may be complex, leading to a possible consequence that the model cannot converge. Second, *workers*, e.g., mobile devices may spend a lot of resources on model training, which affects the response time and ultimately the service-level objective (SLO). When federated learning was first proposed [5], it was assumed that the models are only trained under the condition of plug-in and WiFi connection. However, this statement prevents analyzing geo-spatial data at a fined-grained granularity in a timely manner, violates the ubiquitousness of mobile devices and the purpose of adapting to distributed learning. In addition, a key performance metric of today's machine learning systems [6–9] is realtimeliness. However, mobile devices that train local fresh geo-spatial data in real-time consume large amount of battery energy. When scaling-out, this expensive energy consumption during the training process would increase exponentially, not to mention millions of mobile devices involved, powered by batteries [10]. Therefore, a practical federated learning framework for geo-spatial system should intelligently adapt to privacy issues and take energy efficiency into consideration.

In order to solve the above challenges, it is necessary to analyze and further understand the nature of federated learning. Firstly, We provide research on understanding the state-of-the-practice FL frameworks on several applications. Based on our research, we have drawn the following two important observations: (1) In the history of model training, due to the model complexity and computational cost, we expect the entire framework to start a simple model with a naïve configuration. However, when scaling-out in federated learning, not all *workers* are active with the same degree. Thus, the FL framework may allow *workers* to forget some trained features of the geo-spatial data at any time. In other words, the model version in each *worker* may be different, which is equivalent to a situation where all *workers* are online but not all of their model versions have been updated. This allows us to provide a trade-off when publishing aligned models to *workers* at the beginning of each training round. We can select a subset of *workers* to participate without waking up all devices to reduce energy consumption and maximize the possible reward. The optimization process can reduce the delay and speed up the convergence, thereby reducing local energy consumption. (2) In order to further illustrate the concern about the privacy leak in a federated learning framework, we not only allow each *worker* to delete its sensitive old geo-spatial data, but also allow the model to remove the learned sensitive features. We call this operation "forgets". This technique is often seen as *decremental learning* in the community. This technology can be used in conjunction with the local energy management in the system kernel. When the *worker* is forgetting, we can regard it as a control signal to wake the energy management unit when the computation demand decreases. In this way, we can combine the decremental learning algorithm with previously used system energy management techniques, such as dynamic voltage and frequency scaling (DVFS), process migration, and IC thermal shutdown, so that we can not only improve model training time, but also reduce local energy. These two observations and derived approaches are the key to provide an energy efficient and forget federated learning system that forgets.

According to the aforementioned findings, we present an energy-aware decremental learning framework (SmartDL) which uses decremental learning and energy-saving techniques to well balance the following three factors during the training process incuding: 1) model accuracy, 2) data privacy and 3) energy efficiency. In order to reduce the overall energy consumption from the federated learning, SmartDL serves a two-layered design. When the server creates a learning job, SmartDL triggers an optimization process that models all candidate devices as a multi-armed bandit (MAB) problem, and solves the problem to maximize the target reward (training delay, data volume, and energy consumption). In this way, the entire process can be learned while ensuring performance. When the learning begins, SmartDL designs a local middleware layer, which is based on a specific model and carefully manages the local learning process with incremental and decremental updates. SmartDL intelligently tunes local power state of mobile devices during the decremental learning process. When SmartDL makes a wrong decision in decremental update or prediction, SmartDL can recover the model in corresponding decremental and incremental update algorithm to solve the problem. We have prototyped SmartDL in current mobile operating systems, which supports multiple learning frameworks. The prototype uses machine learning models widely used in real-time machine learning systems for evaluation. The results show that compared with the classic federated learning, SmartDL can reduce energy consumption by 75.6%-82.4% in all workloads. In addition, SmartDL achieves a speedup of 2-4 orders of magnitude in model convergence while ensuring the accuracy of the model. SmartDL can significantly reduce the learning completion time up to 2-4 orders of magnitude, compared to the classic federated learning framework. In all state-of-the-practice baselines, our design shows a 75.6%–82.4% less energy use in all workloads.

The contribution of our work is summarized as follows,

- We reveal the high resource use and privacy leak in the federated learning system on mobile devices for geo-spatial system.
- We propose an energy-aware decremental learning framework (SmartDL) for geo-spatial system, which reduces energy consumption through a two-layer design. The framework performance is provided with a worst-case mathematical guarantee.
- We prototype SmartDL on the physical testbed and simulation platform, and evaluated its performance with several learning benchmarks with real-world traces. Compared with the classic federated learning, SmartDL can save 75.6%-82.4% energy consumption and achieve a speedup of 2-4 orders of magnitude in model convergence while ensuring the accuracy of the model.

The remaining section of the paper is organized as follows. Section 2 introduces the background about our concerns that motivate the design of SmartDL. Section 3 discusses the system design and implementation. Section 4 and 5 present the evaluation and related work, respectively. Then Section 6 concludes the paper.

2 Risen Awareness of Privacy and Resource Use Concern

In this section, we first introduce the working principle of federated learning. Then, we use a real-world example to illustrate the potential privacy leak of federated learning. Finally, we discuss the use of resources in federated learning.

Federated Learning. Federated learning aims to use private data on different edge devices to collaboratively train shared models while protecting data privacy. The specific working principle usually includes the following steps:

1. *Workers* selection: In the initialization phase, the *server* selects a group of mobile candidates (i.e., *workers*) that meet the requirements to participate in the current round of training process.
2. Model broadcast: The *server* broadcasts the current global model status (such as model parameters) to all *workers*.
3. Local training: Each *worker* starts local training with the current state of the shared model and the local data.
4. Parameter aggregation: After completing the local training process, each *work* sends the model parameters to the *server* for parameter aggregation.
5. Model update: After the *server* receives the model parameters sent by *workers*, the *server* updates the model according to the received parameters. After that, the training enters the next round.
6. Model convergence: Iterate the entire training process until the shared model converges.

Note that, the entire training time is not important for each *worker*, because the job can be throttled. However, if the training time of some *workers* is extended, the entire training process may take longer to converge. This is because in each round of training, the original federated learning framework adopts a synchronous mode. In the synchronized state, all training threads keep the device awake, which will generate a lot of energy consumption. Therefore, the training completion time of each *worker* is crucial to the entire process of federal learning, and makes FL very expensive at scale-out.

Privacy in Learning. Here, we use a real-world example to introduce the problem of privacy leak in the process of federated learning, as shown in Figure 1. The *Retailrocket* [11] e-commerce dataset now covers 32,000 de-identified users. It contains some historical records of these users, such as web pages viewed, items added to the shopping cart, and four-month transactions, and so on. Some of these users (such as user A) have repeatedly included the following content in their historical browsing records: *The Godfather, Titanic, Flipped,*

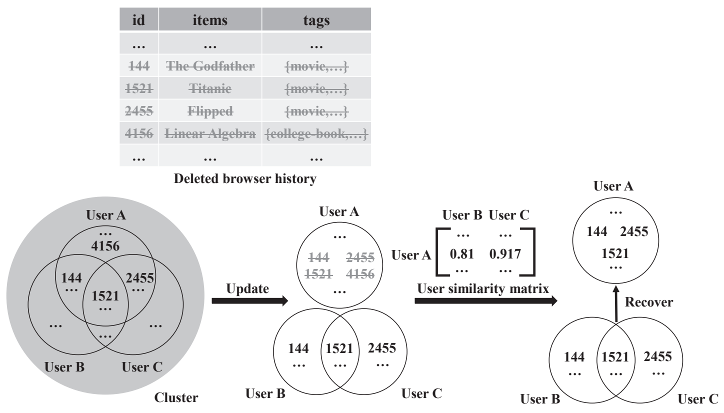


Fig. 1: A possible privacy leak from federated learning, when the original data are deleted. Devices can still reveal the behavior and some privacy guess from user A.

and Linear Algebra. All the information is related to personal privacy, this information is sensitive. Regulations, such as European General Data Protection Regulation (GDPR) [12], give users the right to remove all these sensitive data. However, when user A deletes some records, it only deletes data from data storage such as a database. Federated learning can still reveal private user information based on model clustering. For example, Federated Learning collects some transaction history and browsing records from all users, and calculates a similarity matrix between different users. We can use a simple Jaccard similarity [13] to calculate the similarity distance. This similarity matrix can be used for personal ranked item recommendation. Unfortunately, we can still guess the deleted browsing history of user A. The similarity matrix shown in Figure 1 shows that the average similarity between user A and some users is very high, such as the user B and the user C with a similarity of 0.81 and 0.917, respectively. We can view the undeleted browsing history of user B and user C, and use their information to recover user A's deleted information, *Flipped*, *Titanic* and *The Godfather*. Thus, it can be concluded that we may still obtain these overlapping sets through clustering to reveal personal privacy. Thus, we can find that the existing Federated Learning framework still contains privacy issues.

System Resource Use in Federated Training. In the federated training, there are two main problems, high energy consumption and unnecessary memory usage. As we have already explained in Section 1, all *workers* in the federated learning framework are usually battery-powered edge devices. Previous work [14] has revealed that a federated learning process consumes heavy energy consumption, shortening the overall device service time by 40%. Bonawitz et al. [15] believe that this is only a technical problem, because they assume that the learning models are only trained while the device is plug-in or with WiFi connection. But this assumption violates two important features of federated

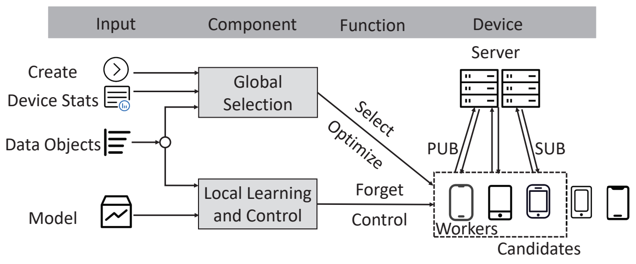


Fig. 2: The system diagram of SmartDL.

learning: (1) freshness of the data objects to be trained; (2) the ubiquity of mobile devices. In addition, that federated learning only happens when plug-in or WiFi connection prevents scale-out. Any unnecessary energy consumption in a single device may affect all other *workers*, leading to exponential energy waste. In addition, during the training process, the learning process needs to retrieve all local data repeatedly from the memory or secondary storage, which can cause a large number of page replacements, and then these page replacements can cause additional delay and unnecessary memory usage. Therefore, in order to solve these existing problems, we need to understand the energy consumption and memory usage in the local training, and design an effective framework that can be used to elastically manage the data in memory, while flexibly selecting enough devices for each training and managing all devices coupled with the energy management policy.

3 Design and Implementation

In the previous section 2, we have discussed the privacy and resource use concerns in the current federated learning framework. In this section, we introduce SmartDL, then analyze our system modeling from the global and local metrics, and finally elaborate global selection optimization and local control with the decremental learning feature.

3.1 System Overview

We show the architecture overview of SmartDL in Figure 2. SmartDL is composed of a global layer and a local layer. The global layer provides a selection and optimization process with the MAB algorithm, while the local layer manages local decremental learning through incremental and decremental updates and the related energy control. The details are as follows.
Global Layer. SmartDL supports federated learning in a client-server manner. In the global selection of the Component as shown in Figure 2, when the server creates a learning job, SmartDL selects a subset of workers $\mathcal{N} = \{1, 2, ...N\}$ from all live candidates based on the device stats. In this subset, all

workers should have required training data D, and have enough computing resources to complete the learning job and get a reward X. In this optimization process, SmartDL will maximize this reward. In the Device as shown in Figure 2, SmartDL initializes the federated learning setup in a PUB/SUB model. The server notifies all selected workers to receive the model to be trained through the PUB method. Gradually, each worker completes its local training and sends model gradients back through SUB methods. Workers can leave at any time during this process. SmartDL allows the server to communicate with workers through the SUB method regularly, and starts the convergence process when the server receives the signal sent back by most of the workers or reaches Time to Live (TTL).
Local layer. In the local learning and control of the Component as shown in Figure 2, each worker introduces a hyperparameter θ, which means how much data one worker should "forget" [16]. Thus, as shown in Figure 1, although we still calculate similarity models between users, SmartDL overwrites the model with newly arrived data and forgets the deleted data, as well as their impact in the model after a certain period of time. In this way, we not only allow the balance between improving the model training time and reducing local energy consumption, but also better protect the data privacy of each worker.

It is worth to note that During the federated learning process, the participating devices consume computing and communication resources. Thus, they are usually reluctant to participate in the training process if the corresponding rewards are not retrieved. Existing work [17] has proposed different incentive mechanisms to motivate devices to actively and reliably participate in the Federated Learning process. These incentive strategies can be directly adopted by SmartDL. Moreover, SmartDL tries to solve a different problem which tries to well balance the following three factors during the training process including: 1) energy efficiency, 2) model accuracy and 3) data privacy. In short, SmartDL shows a two-level design, globally and locally, that optimizes the energy efficiency and privacy in federated learning. Next, we introduce our system modeling in these two layers.

3.2 System Modeling

As shown in Figure 2, our global optimization process assumes that each federated learning job always begins from the server side. Each server receives SUB signals sent by all candidate devices, which contains a profile on the local data volume, available resource, and their battery capacity. We model all variables as global and local metrics respectively. In the global metrics, we find the most suitable subset of one federated learning process based on device statistics, data object information, etc. In the local metrics, we describe how to process local data objects for learning, and model training time and local energy consumption.
Global Metrics. In the global metrics, SmartDL can obtain N mobile devices that participate in the federated learning process, as $\mathcal{N} = \{1, 2, ...N\}$. Each

device $i \in \mathcal{N}$ is related to the reward $X_i(k)$ in round k, where $k = 0,1,2,...$. Specifically, the reward is a normalized variable on $[0, 1]$ and follows a specific distribution with a mean μ_i. As mentioned above, these reward distributions are $i.i.d$, as defined in the basic federated learning framework. Before the whole learning process begins, the mean reward vector $\mu = (\mu_1, ..., \mu_N)$ is unknown.

In each round of learning, due to problems such as network outage, drained batteries, etc., devices can join and leave at any given time. In the PUB/SUB model, any newly arrived device can only subscribe to the next round of learning. In the case of a violation of the TTL $\ddot{T}$ in one learning round, the offline devices are regarded as "sleep". We use $G(k) \in \mathcal{P}(\mathcal{N})$ to denote the set of available mobile devices in a given round k, in which $\mathcal{P}(\mathcal{N})$ is the power set of $\mathcal{N}$. Let $P_G(R) \triangleq P(G(k) = R)$, in which $R \in \mathcal{P}(\mathcal{N})$ represents the distribution of the available devices, which is also i.i.d, as defined in the federated learning framework. Before the start of the training process, the distribution R is also unknown. When the participating devices start to subscribe, SmartDL reveal $G(k)$ on the server side at the beginning of each training round k.

The central server selects m available devices (i.e., these available devices belonging to $G(k)$) in each training round. Each subset of these devices creates a subset of workers. We ensure that the size of the selected set is not greater than m, that is, the server starts to calculate convergence and update models after selecting m workers to publish in order to consider the convergence delay. When We observe the set of available devices R, we use $S(R)$ to represent the set of all feasible groups, that is, $S(R) \triangleq \{S \subseteq R : |S| \leq m\}$, in which $|S|$ represents the cardinality of set S. In a certain training round k, in addition to selecting a group $S(k) \in S(G(k))$, the server also receives a reward represented as $Q(k)$, which is a weighted sum of the reward for each participated device, that is, $Q(k) \triangleq \sum_{i \in S(k)} g_i X_i(k)$, in which g_i is the calculated gradient of device i. We assume that the gradient g_i is a known fixed positive number provided by the models and the upper bound of g_i is represented as $g_{max} > 0$. Besides, the purpose of designing SmartDL is to maximize the expected time-average reward within a given time horizon of K rounds, that is, $\mathbb{E}[\frac{1}{K} \sum_{k=0}^{K-1} Q(k)]$.

Local Metrics. In the local metrics, we first consider the modeling for to-be-trained data objects. Let $l = p(D, \theta)$, where l represents the learnt model, p is a process to learn the model l, $D = \{d_1, d_2, ..., d_n\}$ represents the training data from n devices, and θ is the user-defined coefficient. Now, suppose we need to remove some portion θ of private data d_n of the n-th device. This means that the mdel we actually learn is $l^{'} = p(D \setminus \{d_n\}, \theta)$. This model can easily obtained by repeating the training process p on $(D \setminus \{d_n\}, \theta)$, where θ defines the percentage of attention to the newly created data objects.

Traditionally, retraining a complete model requires retraining all requested data, which may generate a lot of energy consumption. At the same time, the data loss of a single worker (or a small number of workers, not all in the entire set) does not significantly change the predictability of the model. So we update the existing model l to the desired model $l^{'}$, or update the process p to the process p_{forget}. This "forget" process p_{forget} can check d_n much faster than retrained p, and meanwhile it can still converge, as follows:

$$l^{'} = p_{forget}(p(D,\theta), \{d_n\}, \theta) = p(D \setminus \{d_n\}, \theta) \tag{1}$$

This methodology is called *decremental learning* [18], similar to online learning. But in online learning, we only use new observed data to incrementally update the existing model. In decremental learning, in addition to using new observed data to update incrementally, we also need to remove such updates via the reverse operation.

In SmartDL, we consider the actual scenario when federated learning is deployed to the real devices, and mainly focus on the energy and latency from local training and modeling. In order to understand the training time and the estimated power consumption from one local training, we adopt models from previous research [14].

For the completion time of one local training, the time required to complete a round of training on different devices is different for a specific training task. This is because the training completion time can be determined by three factors: the size of the data objects, the training platforms, and the hardware configuration of the mobile devices. Specifically, 1) the size of the data object d_i, this is because different mobile devices can have different numbers of local data objects. 2) Training platforms γ, this is because different clients can have different training platforms. 3) Hardware configuration of the mobile devices (eg., number of cores c, core frequency f). This is because the hardware configuration of the mobile device may be quite different even for the same training platform. Therefore, we model the completion time of one local training as follows:

$$t_i = \frac{\gamma * d_i}{c^{\alpha} f^{\beta}} \tag{2}$$

where γ is a coefficient, which represents different training platforms. d_i is the number of the data objects required for one round of training. c is the number of cpu cores used in the training process. f is the core frequency. α and β are the corresponding coefficients that can be obtained through the system identification process. We find that the training completion time is directly proportional to the size of the data objects, and inversely proportional to the number of cpu cores and core frequency.

For the power consumption of one local training, the power consumption can vary according to the number of cores and cpu frequency. Therefore, we model the power consumption of a local training as follows:

$$P_i = P_{BL,N_c} + \sum_{i}^{N_c} P_{core,U_i,f_i} \tag{3}$$

N_c represents the number of cores enabled, P_{BL,N_c} represents the baseline CPU power when N_c cores are enabled, and P_{core,U_i,f_i} represents power increment of core i when working at frequency f_i with utilization U_i. In addition, we model the single core power P_{core,U_i,f_i} as follows:

$$P_{core,U_i,f_i} = \zeta * f_i * U_i + \zeta_{base} \tag{4}$$

in which, ζ is a coefficient of the corresponding power.

Thus, the energy consumption required to complete the overall training is can summarized as the following equation:

$$e = \int^T f_{CPU}\bar{U} + \sum_j e_j \tag{5}$$

where $\bar{U}$ is the average utilization. f_{CPU} is an energy coefficient with a given frequency. T is the training completion time, e_j is a static energy profile for each other devices on their specific power states, based on the state-of-the-art state-machine models [19,20]. The energy consumption e is a linear combination of device utilization and energy states.

Next, we model the completion time required to complete the overall training. equation (2) reveals a linear correlation between the size of local training data under certain model specifications. Therefore, the model for training completion time is now simplified as follows:

$$T = A * F(w, \mathcal{M}, D) + B, \text{where } f_j \text{ is fixed} \tag{6}$$

where, in each round of training, the completion time is positively correlated to a function F of the priority weight w, local model $\mathcal{M}$, and affected data size D, under the current CPU frequency f_j in each round of training. A and B are correlation metrics. For all the above models, SmartDL can provide a decremental learning version of local training based on the data and performance (i.e., energy and latency models). Please note that We need to do corresponding work for each type of specific model. In addition, SmartDL may choose the wrong devices due to a prediction error. SmartDL can only fix the wrong choice when another federated learning job initializes. The results indicate that this may only affect the 95%-percentile performance. On average, compared to other state-of-the practice, SmartDL can sustain a better federated learning service. More discussion on fault tolerance can be found in Section 4.

3.3 Global Selection Optimization

We use a binary vector $B(k) = (b_1(k), ..., b_N(k))$ to represent whether a certain device is selected to participate in the training in round k to describe the decision of worker selection. $b_i(k) = 1$ if device i is selected, then $i \in S(k)$; otherwise, $b_i(k) = 0$. Thus, for all $k \geq 0$, the action vector $B(k)$ must satisfy $\sum_{i=1}^{N} b_i(k) \leq m$.

Assuming that a vector of mean reward μ is known in advance, we can always formulate the reward maximization in federated learning with minimum selection fraction as an optimization problem [21]:

$$\begin{aligned}
\text{Maximize } & \sum_{R\in\mathcal{P}(\mathcal{N})} P_G(R) \sum_{S\in S(R)} b_S(R) \sum_{i\in S} g_i\mu_i \\
\text{subject to: } & \sum_{R\in\mathcal{P}(\mathcal{N})} P_G(R) \sum_{S\in S(R):i\in S} b_S(R) \geq r_i, i\in N \\
& p_i(R) \leq B_i \in [0,1], R\in P(N)
\end{aligned} \tag{7}$$

But the reward vector μ is unknown, so SmartDL not only needs to use the estimated mean rewards (i.e., exploitation) to maximize the reward, but also has to learn to obtain a more precise estimate of the mean rewards (i.e., exploration) to maximize the reward at the same time. In this way, we chose an online optimization algorithm to deal with the exploitation and exploration tradeoff problem (EE problem).

We treat the SmartDL selection optimization as an MAB problem. MAB problem considers a fixed limited set of resources to be allocated between competing (alternative) choices in a way that maximizes their expected gain, when each choice's properties are only partially known at the time of allocation [22]. In the multi-armed bandit problem, the main challenges we focus on are how to maximize the reward with unknown mean reward distribution and out-of-date worker, and how to minimize the energy consumption without violating the TTL constraint.

In the case of uncertainty, the first key point to maximize the reward is to quickly retrieve the reward distribution from top workers. We use $c_i(k)$ to represent the number of times worker i selected at the end of round k, that is, $c_i(k) \triangleq \sum_{t=0}^{k} b_i(t)$. When the system begins at $k=0$, We set $c_i(-1)=0$. At the same time, let $\widehat{\mu}_i(k)$ be the observed mean rewards of worker i by the end of round k, i.e., $\widehat{\mu}_i(k) \triangleq \frac{\sum_{t=0}^{k} X_i(t)b_i(t)}{c_i(k)}$. If worker i has not been selected before the end of round k (i.e., if $c_i(k)=0$), we set $\widehat{\mu}_i(k)=1$. We use $\bar{\mu}_i(k)$ to represent the estimation of worker i in round k, which is given as follows:

$$\bar{\mu}_i(k) \triangleq \min\{\widehat{\mu}_i(k-1) + \sqrt{\frac{3\log k}{2c_i(k-1)}}, 1\} \tag{8}$$

where $\widehat{\mu}_i(k-1)$ and $\sqrt{\frac{3\log k}{2c_i(k-1)}}$ correspond to exploitation and exploration, respectively. Because the actual reward must be [0,1], the upper limit of the above truncated version of the reward estimate is 1. Similarly, if $c_i(k-1)=0$, then $\bar{\mu}_i(k)=1$. The detailed optimization analysis has been elaborated in our previous work [23]. It is worth to note that the training environment of federated learning is highly dynamic. For instance, the mobile devices can randomly shutdown due to different issues such as out-of-battery or system failure. The network condition is also not stable during the training process. In addition, the devices can leave or join the federation at any time. Thus, the possibility that a specific device always has the lowest reward score is relatively low in real-world scenario. Next, we introduce our local learning and control to analyze the local decremental learning and energy control in detail.

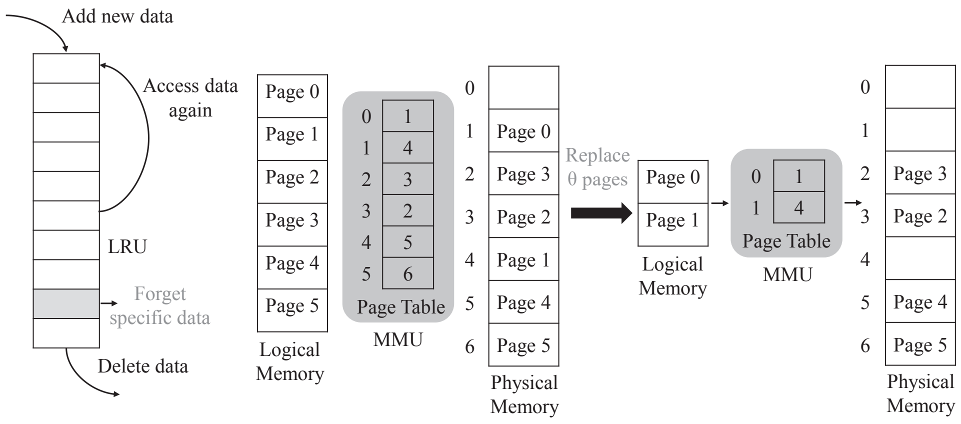


Fig. 3: θ-LRU Memory Management.

3.4 Local Learning and Control

In order to solve the previously described privacy leak and heavy resource usage issues, We use a local decremental learning, which respects the maximum rewards obtained by the selected workers. The main idea of our method is that the training algorithm of the target model retains the intermediate results during model calculation. We can effectively update the intermediate in two ways: incremental method to merge new user data, and decremental method to delete user data. As such, we are able to fine tune the local energy configuration, as well as provide a more accurate profile of the local device for next round of global selection.

We use the following four aspects to illustrate that SmartDL adapts learning algorithms into local training:

– *Model Construction*: Construct the prediction model based on the characteristic of the specific learning algorithm.
– *Update Procedure*: Design the corresponding decremental and incremental update algorithms based on the model established in *Model Construction*, and save energy through the dynamic voltage and frequency scaling (DVFS) tuning function.
– *Space Complexity*: Analyze the space and complexity of decremental and incremental updating strategy of the design, with a θ-LRU memory management.
– *Data Recovery*: Analyze how to recover deleted user data from the stale model.

Memory Management. To adapt a specific learning algorithm into the local SmartDL middleware, SmartDL first rebuilds this algorithm into a decremental version. This process is usually done offline, and SmartDL explores this decremental learning algorithm as a local learning algorithm library. The decremental learning algorithms focus on the incremental/decremental up-

Algorithm 1 Update procedure for Personalized PageRank.

```
Input: history matrix $\mathbf{Y}_u$, similarity matrix $\mathbf{L}$, concurrency matrix $\mathbf{C}$, item interaction count vector $\mathbf{v}$
 1: function UPDATE($\mathbf{Y}_u, \mathbf{L}, \mathbf{C}, \mathbf{v}$)
 2:     $\mathbf{v} \leftarrow \mathbf{v} + \mathbf{Y}_u$
 3:     while item pair $(i_1, i_2) \in \mathbf{Y}_u$ do
 4:         $C_{i_1 i_2} \leftarrow C_{i_1 i_2} + 1$
 5:     while item $i_1 \in \mathbf{Y}_u$ do
 6:         while item $i_2 \in \mathbf{C}_{i_1}$ do
 7:             $L_{i_1 i_2} \leftarrow C_{i_1 i_2} / (v_{i_1} + v_{i_2} - C_{i_1 i_2})$
 8:             CPU_Freq(1)  //Tune up DVFS
 9: function FORGET($\mathbf{Y}_u, \mathbf{L}, \mathbf{C}, \mathbf{v}$)
10:     $\mathbf{v} \leftarrow \mathbf{v} - \mathbf{Y}_u$
11:     while item pair $(i_1, i_2) \in \mathbf{Y}_u$ do
12:         $C_{i_1 i_2} \leftarrow C_{i_1 i_2} - 1$
13:         CPU_Freq(-1)  //Tune down DVFS
14:     while item $i_1 \in \mathbf{Y}_u$ do
15:         while item $i_2 \in \mathbf{C}_{i_1}$ do
16:             $L_{i_1 i_2} \leftarrow C_{i_1 i_2} / (v_{i_1} + v_{i_2} - C_{i_1 i_2})$
17:             CPU_Freq(0)  //Reset DVFS
18: function PREDICT($k, j, \mathbf{L}$)
19:     return top-$k$ items from $\mathbf{L}_j$
```

dates, associated with local energy control. Moreover, in order to reduce the frequency of page replacement in local decremental learning, we adopt a new θ-LRU memory management as shown in Figure 3. LRU (Least Recently Used) [24] is a commonly used page replacement algorithm. It is based on the idea of "if a certain data has not been accessed in the recent period, then the possibility of it being accessed in the future is also very small". The LRU algorithm selects the most recently unused pages and eliminates them. However, When being notified with a degree of "forget", or the user-defined variable θ, the SmartDL middleware adapts a θ-LRU as shown in Figure 3, that deletes the specific user data and only replaces θ-percent of allocated pages recently used. This algorithm can significantly reduce the frequency of page replacement, as well as the number of swaps. If the replaced pages are not an integer, we use the method of rounding up. SmartDL keeps track of the level of forgetness in the decremental learning algorithms using data recovery policies, in order to prevent aggressive forgetting and the convergence failure. Next, we present SmartDL on three learning algorithm cases, namely Personalized PageRank, Tikhonov Regularization and K-Nearest Neighbors, which are widely used in the real-time mobile-based machine learning, in order to highlight the design and implementation of SmartDL in the local layer, and show that SmartDL can be easily adapted to effectively support other algorithms and systems.

Case 1: Personalized PageRank. Personalized PageRank (PPR) [25] is the first algorithm proposed by Google that uses simple hyperlinks to calculate the scores of web pages, thereby ranking web pages. This algorithm is similar to the recommendation algorithm [26–29]. They both calculate the distance be-

tween users from the perspective of user-item correlation. For example, the algorithm records the browsing history of each device in web page recommendation. These pairs of co-occurring pages are ranked and formed the basis for recommendation later. The input data of the simplest form for PPR includes a binary history matrix $\mathbf{Y} \in \{0,1\}^{|A|\times|I|}$, which represents the interactions between a set of devices A and a set of items I. If the device j interacts with the item i, the entry Y_{ji} is equal to 1, otherwise it is equal to 0.
Model Construction. Personalized PageRank is composed of a similarity matrix $\mathbf{L} \in \mathbb{R}^{|I|\times|I|}$, which denotes the interaction similarity between pairs of items. A common training method for this model is to first compute the concurrency matrix $\mathbf{C} = \mathbf{Y}^{\mathrm{T}}\mathbf{Y}$, which denotes the number of users interacting with each pair of items. Besides, we need a vector $\mathbf{v} = \sum_{j\in A} \mathbf{Y}_j$ to denote the number of interactions for each item (the sum of the rows of $\mathbf{Y}$). Next, when we need to get the similarity matrix $\mathbf{L}$, we can calculate it by checking the co-occurrence counts. The co-occurrence matrix [30] can calculate many similarity measures between items. It is a better choice to calculate the Jaccard similarity $L_{i_1 i_2}$ between items i_1 and i_2 by $L_{i_1 i_2} = C_{i_1 i_2}/(v_{i_1} + v_{i_2} - C_{i_1 i_2})$. The results can be achieved by querying the similarity matrix $\mathbf{L}$, as described in the example of privacy issue in Figure 1. As shown in the PREDICT function (line 19) in Algorithm 1, we retrieve recommendations of item pairs by querying the most similar items in each item, and calculate the preference estimates based on the weighted sum between the similarities of the item and the corresponding user history to generate the items to recommend for specific devices [31].
Update Procedure. We need the following three intermediate data structures including: 1) the similarity matrix $\mathbf{L}$, 2) the concurrency matrix $\mathbf{C}$ and 3) the item interaction count vector $\mathbf{v}$, to enable the incremental and decremental updates for Personalized PageRank.

The FORGET function (lines 10-17) of Algorithm 1 details the entire process of removing the u-th user data (corresponding to the u-th row $\mathbf{Y}_u$ in the history matrix $\mathbf{Y}$). We update the corresponding co-occurrence count $C_{i_1 i_2}$ by traversing all item pairs (i_1, i_2) in the user history $\mathbf{Y}_u$. Finally, we need to traverse each item in the user history and renew the similarity matrix in corresponding row $\mathbf{L}_{i_1}$. The working principle of the incremental update of the model, explained in the UPDATE function (lines 2-8) of Algorithm 1 is similar, the difference is that we increase the co-occurrence count instead of reducing the co-occurrence count.
Space Complexity. Personalized PageRank is composed of a similarity matrix $\mathbf{L} \in \mathbb{R}^{|I|\times|I|}$ with the space quadratic of the number of items. Since we need to protect the concurrency matrix $\mathbf{C} \in \mathbb{V}^{|I|\times|I|}$ and the vector $\mathbf{v} \in \mathbb{V}^{|I|}$, we need to adjust $|\mathbf{Y}_u|^2$ of the concurrency matrix, and the $|\mathbf{Y}_u| \cdot |I|$ recalculation entries in the similarity matrix $\mathbf{L}$ in the worst case. The intermediate data structure of the decremental learning algorithm double the required memory, and the update has a quadratic complexity of $O(|I|^2)$ in the worst case. In practice, for example, given a $\theta = 30\%$ configuration and PPR on $I = 1000$ items, SmartDL uses θ-LRU to reduce up to 378 page swaps in memory replacement during a single round. However, most users need to interact with

Algorithm 2 Update procedure for Tikhonov Regularization.

```
Input: $\mathbf{z} \leftarrow \mathbf{M}^{\mathrm{T}}\mathbf{r}, \mathbf{QR} \leftarrow qr(\mathbf{M}^{\mathrm{T}}\mathbf{M} + \lambda\mathbf{I})$, user observation $\mathbf{M}_u$
 1: function UPDATE($\mathbf{M}_u, r_u, \mathbf{Q}, \mathbf{R}, \mathbf{z}$)
 2:     $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{M}_u r_u$
 3:     $\mathbf{QR} \leftarrow qr_{update}(\mathbf{Q}, \mathbf{R}, \mathbf{Q}^{\mathrm{T}}\mathbf{M}_u, \mathbf{M}_u)$
 4:     solve $\mathbf{Rh} = \mathbf{Q}^{\mathrm{T}}\mathbf{z}$ for $\mathbf{h}$
 5:     CPU_Freq(1)
 6: function FORGET($\mathbf{M}_u, r_u, \mathbf{Q}, \mathbf{R}, \mathbf{z}$)
 7:     $\mathbf{z} \leftarrow \mathbf{z} - \mathbf{M}_u r_u$
 8:     $\mathbf{QR} \leftarrow qr_{update}(\mathbf{Q}, \mathbf{R}, -\mathbf{Q}^{\mathrm{T}}\mathbf{M}_u, \mathbf{M}_u)$
 9:     solve $\mathbf{Rh} = \mathbf{Q}^{\mathrm{T}}\mathbf{z}$ for $\mathbf{h}$
10:     CPU_Freq(-1)
11: function PREDICT($\mathbf{m}_{new}, \mathbf{h}$)
12:     return $\mathbf{h}^{\mathrm{T}}\mathbf{m}_{new}$
```

very few items in the real world data. In addition, we only retain the top-k entries of each item in $\mathbf{L}$. At the same time, we introduce bounds on the maximum number of interactions to reduce the memory usage required for the intermediate data structures and the update complexity [16]. The number of energy control function calls is linear to the number of function calls for incremental/decremental updates. Although SmartDL uses the similarity matrix to find corresponding users, it may selectively reduce the data. As the training proceeds, the new data overwrite the old data. Moreover, the new data could be detected after a few training rounds. The detailed analysis is discussed in Section 4.

Data Recovery. We analyze how to recover deleted user data from the stale model by deleting individual user data from the database. When the original matrix $\mathbf{Y}$ deletes the row corresponding to the removed user, the matrix $\mathbf{Y}$ becomes an updated matrix $\hat{\mathbf{Y}}$. If we still have access to the similarity matrix $\mathbf{L}$ calculated from the original matrix $\mathbf{Y}$, then we can compute the corresponding similarity matrix $\hat{\mathbf{L}}$ from the updated matrix $\hat{\mathbf{Y}}$ and compare it with the stale similarity matrix $\mathbf{L}$. All items i with differences in entries of the similarity matrices (e.g., $\exists j\ \mathbf{L}_{ij} \neq \hat{\mathbf{L}}_{ij}$) are exactly the items that were included in the interaction history of the deleted device. In this way we can recover the deleted data.

Case 2: Tikhonov Regularization. Tikhonov regularization [32] is a technique widely used to analyze multiple regression data that suffer from multicollinearity. The input data of this model is the matrix $\mathbf{M} \in \mathbb{R}^{s \times d}$ of s d-dimensional observations, and the corresponding digital target variable $\mathbf{r} \in \mathbb{R}^s$. Under the principle of ensuring generality, we assume that the data of a specific user i is captured in the input i-th row vector $\mathbf{M}_i$. If there is more than one row of data representing a particular user, we can simply perform the decremental update process several times.

Model Construction. The solution of the normal equation $\mathbf{h} = (\mathbf{M}^{\mathrm{T}}\mathbf{M} + \lambda\mathbf{I})^{-1}\mathbf{M}^{\mathrm{T}}\mathbf{r}$ is a common method to calculate the Tikhonov regularization model in the form of the weight vector $\mathbf{h}$. As shown in the PREDICT function

(line 12) of Algorithm 2, we can use the weight vector as the dot product to calculate the estimate of a new observation $\mathbf{m}_{new}$: $\hat{r}_{new} = \mathbf{h}^{\mathrm{T}}\mathbf{m}_{new}$.
Update Procedure. We use an effective calculation method to explain the process of deleting the data of a certain device u (corresponding to the u-th row $\mathbf{M}_u$ of matrix $\mathbf{M}$) from the Tikhonov regularization model:

$$\mathbf{h} = (\mathbf{M}^{\mathrm{T}}\mathbf{M} - \mathbf{M}_u^{\mathrm{T}}\mathbf{M}_u + \lambda\mathbf{I})^{-1}(\mathbf{M}^{\mathrm{T}}\mathbf{r} - \mathbf{M}_u r_u) \tag{9}$$

In this way, we retain two intermediates in the calculation, namely the vector $\mathbf{z} = \mathbf{M}^{\mathrm{T}}\mathbf{r}$ and a QR factorization $\mathbf{QR} = qr(\mathbf{M}^{\mathrm{T}}\mathbf{M}+\lambda\mathbf{I})$ of the regularized gram matrix. First, we have to use the method of subtracting $\mathbf{M}_u r_u$ to recalculate $\mathbf{z}$, and we have to update the QR decomposition $\mathbf{Q}$ and $\mathbf{R}$ through using the fast rank-one update algorithm [33] with $-\mathbf{Q}^{\mathrm{T}}\mathbf{M}_u$ and $\mathbf{M}_u$ as parameters. Then, we can use the FORGET function (lines 7-10) of Algorithm 2 to solve the updated model $\mathbf{h}$. If we use addition instead of subtraction, we can get an incremental update algorithm, as shown in the UPDATE function (lines 2-5) of Algorithm 2.
Space Complexity. We need to maintain two additional matrices $\mathbf{Q} \in \mathbb{R}^{d\times d}$ and $\mathbf{R} \in \mathbb{R}^{d\times d}$ as well as the vector $\mathbf{z} \in \mathbb{R}^d$ in the decremental variant of the model. This means that the space required for Tikhonov regularization is quadratic in the number of feature number d and has nothing to do with the number of examples. It is usually much less than the number of examples s. An decremental/incremental update requires scaling and adding to $\mathbf{z}$ ($2d$ operations), the matrix vector multiplication $\mathbf{Q}^{\mathrm{T}}\mathbf{M}_u$ (d^2 operations), the rank-one QR update [33] ($26d^2$ operations), the matrix vector multiplication $\mathbf{Q}^{\mathrm{T}}\mathbf{z}$ (d^2 operations), and solving for $\mathbf{h}$ by reverse substitution (d^2 operations). SmartDL introduced the FORGET function (lines 7-10) of Algorithm 2. Therefore, in this linear correlated algorithm, θ-LRU can significantly reduce more page errors. All in all, the complexity of our update is $O(d^2)$, which improves from the original retraining $O(sd^2)$ complexity.
Data Recovery. Compared with other methods, it is more difficult to obtain information about the deleted device feature vector $\mathbf{M}_d$ from the model $\mathbf{h}$. Although we can constrain the candidate vectors in the subspace defined by $\mathbf{h}^{\mathrm{T}}\mathbf{M}_d = r_d$ via accessing the complete target variable $\mathbf{r}$, this can generate a lot of prediction errors, so we need to know more about $\mathbf{M}$ to further control the candidate vector space.
Case 3: K-Nearest Neighbors. K-Nearest Neighbors (K-NN) [34] is a statistical algorithm for classification, which assigns the labels of the k nearest neighbors to the unknown observations. In this case, we use a general method, locality sensitive hashing (LSH) [35], to speed up the search for the nearest neighbors. The input data of this model includes a matrix $\mathbf{M} \in \mathbb{R}^{s\times d}$ of s d-dimensional observations, and the corresponding target variable $\mathbf{r} \in \{1, ..., a\}^s$ which represents the assignments of the corresponding a categorical labels. We again declare that the i-th row vector $\mathbf{M}_i$ corresponds to the observation of a specific user i.
Model Construction. The algorithm builds an index on the data and uses the approximate similarity of the high-dimensional space to search. This index

Algorithm 3 Update procedure for K-Nearest Neighbors.

```
Input: hash tables $E_1, ..., E_h$, projection matrices $\Psi_1, ..., \Psi_h$, observation $\mathbf{M}_u$
 1: function UPDATE($\mathbf{M}_u, \Psi, E$)
 2:     while $h = 1...H$ do
 3:         $y_{uh} = \text{sgn}(\mathbf{M}_u \Psi_h)$
 4:         add $\mathbf{M}_u$ from bucket $y_{uh}$ in hash table $E_h$
 5:         CPU_Freq(1)
 6: function FORGET($\mathbf{M}_u, \Psi, E$)
 7:     while $h = 1...H$ do
 8:         $y_{uh} = \text{sgn}(\mathbf{M}_u \Psi_h)$
 9:         remove $\mathbf{M}_u$ from bucket $y_{uh}$ in hash table $E_h$
10:         CPU_Freq(-1)
11: function PREDICT($\mathbf{M}_{new}, \Psi, E$)
12:     $U \leftarrow$ a binary heap with the $k$ closest examples
13:     while $h = 1...H$ do
14:         $y_{uh} = \text{sgn}(\mathbf{M}_u \Psi_h)$
15:         update $U$ for all examples from bucket $y_{uh}$
16:     return majority label from $k$ examples in $U$
```

consists of several hash tables, among which the observations that are close according to Euclidean distance are likely to eventually appear in the same hash bucket. We calculate the bucket index by using random projections as hash function. We calculate the H hash table $E_1, ..., E_H$ by using the y-dimensional bucket indexes. We separately generate a Gaussian random matrix Ψ_h according to each hash table E_h, and use sgn($\mathbf{M}\Psi_h$) to make this matrix randomly project our data and the resulting bit vectors represent the hash keys. We use the key sgn($\mathbf{M}_i\Psi_h$) in the hash table E_h to assign each row of $\mathbf{M}_i$ from $\mathbf{M}$ to its corresponding bucket. For each hash table E_h, we calculate the bucket index $y_h = \text{sgn}(\mathbf{M}_{new}\Psi_h)$ and collect all observations from this bucket to collect its nearest neighbors, thereby assigning labels to the invisible observation $\mathbf{M}_{new}$. Then, we calculate the k nearest neighbors of $\mathbf{M}_{new}$ among the retrieved observations, and assign most of the labels to $\mathbf{M}_{new}$ as the predicted label $\hat{l}_{new}$. As shown in the PREDICT function (lines 12-16) of Algorithm 3, this can be effectively achieved by using the k closest examples to maintain the binary heap.

Update Procedure. As described in the FORGET function (lines 7-10) of Algorithm 3, deleting the existing observation $\mathbf{M}_u$ of a user u from our index works. We calculate the bucket index y_{uh} of $\mathbf{M}_u$ by the random projection sgn($\mathbf{M}_u\Psi_h$), and delete $\mathbf{M}_u$ from bucket y_{uh} in hash table E_h. We repeat this process for all hash tables. The incremental update works in a similar way, as shown in the UPDATE function (lines 2-5) of Algorithm 3.

Space Complexity. Our method does not need extra space, because it only uses the hash tables $E_1, ..., E_h$ and the projection matrices $\Psi_1, ..., \Psi_h$, which are always necessary to derive predictions from the model. The complexity of the decremental update is $O(Hdy)$, because it obtains a projection matrix of size $d \times y$ and the feature vector of dimensionality d, and performs H matrix-vector

Table 1: Hardware Information of Mobile Devices

Device	Android Version	Number of CPU cores	Maximum CPU Frequency
Honor 8 Lite	8.0.0	8	2.112GHz
Nexus 6	5.0.0	4	2.700GHz
Lenovo	5.0.2	4	1.040GHz
ZTE	5.1.1	4	1.094GHz
Xiaomi	5.1.1	6	1.440GHz

multiplication of the two to determine the bucket indices, plus H subsequent hashmap insertion.
Data Recovery. It is very simple to recover removed data from the model, this is because that the model stores a copy of the feature vector.

It is worth to note that. SmartDL may take some effort to change decremental and incremental updates for a specific case. However, SmartDL can provide the design and evaluation of the representative models that are commonly used which can cover most of the usage cases. Moreover, before the overall training process starts, the task publisher can directly adopt an existing or design a new corresponding decremental/incremental strategy and then triggers the learning procedure. After that, SmartDL can effectively manage the system in order to achieve high energy efficiency and privacy protection.

4 Evaluation

SmartDL is designed to intelligently balance the energy efficiency and data privacy for on-device federated learning while guaranteeing the model accuracy. In this section, we introduce the experimental setup and then discuss the corresponding evaluation results in detail from different perspectives.

4.1 Experimental Setup

We evaluate the effectiveness of SmartDL with the following two platforms: physical testbed and simulation. For the physical testbed, we prototype SmartDL based on mobile devices with different hardware configurations. Specifically, Table 1 shows the hardware information of the mobile devices adopted in the experiments. The on-device training process is implemented based on the deep learning framework DL4J [36]. Moreover, a Monsoon Power Monitor [37] is used to measure the power consumption of the participating devices during the local training process. In addition, for the scalability evaluation, we also establish a simulation platform. Specifically, we use various independent docker images to emulate different mobile devices, and simulate the Federated Learning training environment by deploying hundreds of corresponding

FL docker images. The docker images of the corresponding experiments are provided on DockerHub [1].
Baselines: We use the following two baselines to compare with SmartDL in the evaluation process.

- **Original:** On the server side, model aggregation is conducted when all the participating devices complete the local training process. Each participated mobile device leverages all the training data within it for local training without performing incremental or decremental learning.
- **NewFL:** On the server side, NewFL adopts the same aggregation approach as Original. However, on the device side, only the newly generated data objects are used in the local training process.

Models and Datasets: The following models and datasets are adopted in order to evaluate the effectiveness of SmartDL in different scenarios. For the comparison with Original, we use the following four models including: 1) Personalized PageRank, 2) K-Nearest Neighbors, 3) Multinomial Naïve Bayes and 4) Tikhonov Regularization. In addition, we evaluate the effectiveness of SmartDL on different models using different datasets. Specifically, the description of the datasets for the corresponding models is as follows:

- Personalized PageRank: 1) movie ratings (movielens) and 2) the joke ratings (jester) [38].
- K-Nearest Neighbors and Multinomial Naïve Bayes: 1) mushrooms classification (mushrooms), 2) puishing websites (puishing) and 3) cartographic forest data (covtype) [39].
- Thikonov Regularization: 1) housing prices prediction (housing, cadata), 2) music year prediction (YearPredictionMSD) [39].

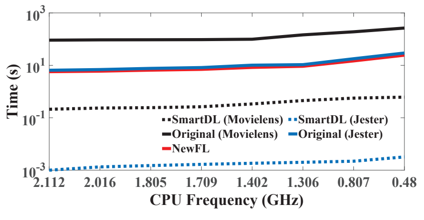
Moreover, for the comparison with NewFL, we use a CNN network which contains a convolutional layer and a fully connected layer based on the Cifar-10 dataset [40]. It is important to note that It is important to note that there are existing works about the proof of convergence of federated learning and decremental learning [41]. These existing methodologies can be applied to conduct the convergence analysis in SmartDL. For the privacy leakage avoidance proof. We will try to conduct it in our future work. Moreover, in order to evaluate the effectiveness of SmartDL on convergence guarantee and privacy leakage avoidance, we conducted different experiments in the evaluation section (e.g., Figure 6 shows the model accuracy of SmartDL and the original federated learning scheme. Figure 9 shows the privacy protection result of different schemes.).

4.2 Results

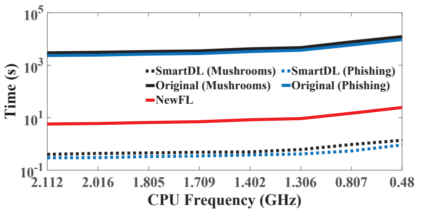
4.2.1 Comparision of Training Completion Time

We conduct the local training process on off-the-shelf mobile devices and evaluate the training completion time of different schemes on different models with

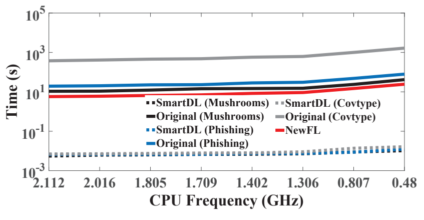
[1] https://hub.docker.com/r/goodlab/deal



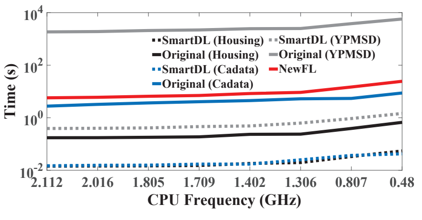
(a) Personalized PageRank.



(b) K-Nearest Neighbors.



(c) Multinomial Naïve Bayes.



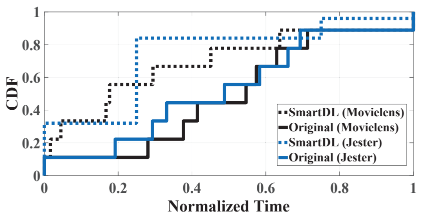
(d) Tikhonov Regularization.

Fig. 4: Training completion time of different application scenarios with different schemes.

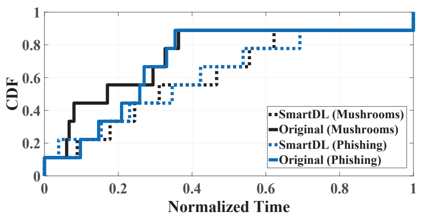
various datasets. Specifically, for each experiment, we randomly select twenty users and report the average training completion time in different scenarios. Figure 4 shows the experimental results under different CPU frequencies on Huawei Honor 8 Lite.

Specifically, Figure 4(a) shows the results of training completion time of Personalized PageRank model on the movielens and jester datasets. In the movielens datasets, SmartDL achieves one and two orders of magnitude faster than NewFL and Original, respectively. This is because SmartDL not only trains less data than Original, but also SmartDL forgets some updates during training process, thus greatly reducing the overall training completion time. Similarly, Figure 4(b) shows the results of K-Nearest Neighbor model on the mushrooms and phishing datasets. SmartDL has the similar advantage on the training completion time, which is one order of magnitude and three orders of magnitude faster than NewFL and Original, respectively. In addition, when we allow to use aggressive DVFS on the mobile device, SmartDL in the phishing dataset can achieve four orders of magnitude better performance than Original. This is mainly due to the following two reasons. First, the phishing dataset is very large, and during the overall training process, SmartDL has higher I/O intensity than other baselines, which results in higher power saving potential. Second, original always trains the full dataset, The memory footprint of the dataset is much larger than SmartDL.

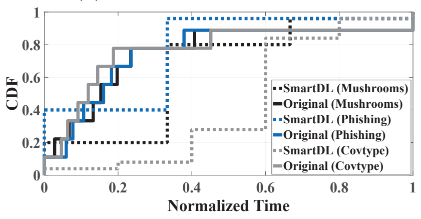
In the other two cases, Figures 4(c) and 4(d) both show that compared with the two baselines, SmartDL can achieve two to four orders of magnitude faster in training completion time. However, the performance of SmartDL slowly converges to that of NewFL when the data are more coarse-grained such as the YearPredictionMSD dataset. Yet, SmartDL always achieves a better



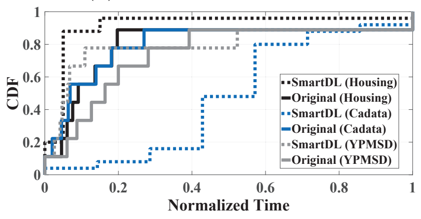
(a) Personalized PageRank.



(b) K-Nearest Neighbors.



(c) Multinomial Naïve Bayes.



(d) Tikhonov Regularization.

Fig. 5: CDF of the convergence time of SmartDL and Original in different application scenarios with default governor.

modeling performance than NewFL because it can capture feature from older data objects.

4.2.2 Comparision of Convergence Time

We deploy hundreds of FL docker images to simulate the participated mobile devices and the corresponding training environment. Figure 5 shows the convergence time of SmartDL and Original in different application scenarios with default governor. The CDF result of Figure 5(a) shows the trend of the convergence time of SmartDL and Original on Personalized PageRank model with the default power governor (interactive). Concretely, it can be seen that in the Movielens dataset, 92% of the simulated devices take shorter time to converge with SmartDL than the Original scheme. The median values of convergence time for SmartDL and Original are 158ms and 94,988ms (normalized to 0.18 and 0.55) respectively. For the Jester dataset, 85% of the simulated devices show that SmartDL converges faster compared with Original. The median value of the convergence time for SmartDL and Original are 1ms and 6,598ms (normalized to 0.25 and 0.36) in this case.

The CDF result of Figure 5(b) also shows the similar result. In the Mushrooms dataset, the median value of the convergence time for SmartDL and Original are 400ms and 2,931,949ms (normalized to 0.31 and 0.17). In the Phishing dataset, the median value of the convergence time for SmartDL and Original are 277ms and 2,403,335ms (normalized to 0.35 and 0.26). It shows that in the K-Nearest Neighbors model, the convergence time of SmartDL of more than half of the mobile devices is four orders of magnitude faster than Original.

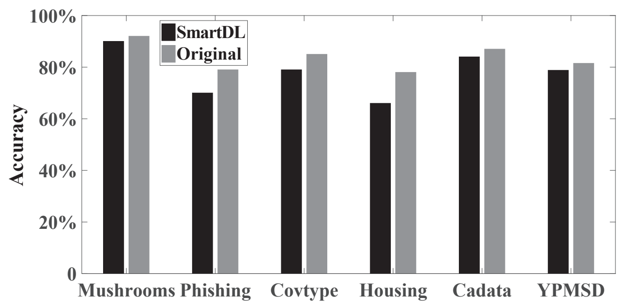


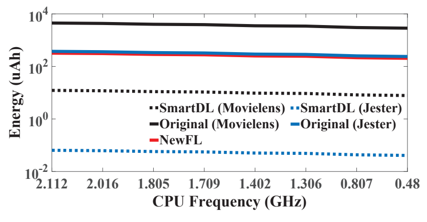
Fig. 6: Accuracy of SmartDL and Original on Tikhonov Regularization Model with different datasets.

The CDF result of Figure 5(c) shows the trend of the convergence time of SmartDL and Original on the Multinomial Naïve Bayes model. Specifically, in the Mushrooms dataset, the median value of the convergence time for SmartDL and Original are 5ms and 9,365ms(normalized to 0.3 and 0.13). In the Phishing dataset, the median value of the convergence time for SmartDL and Original are 6ms and 19,317ms (normalized to 0.33 and 0.14). In the Covtype dataset, the median value of the convergence time for SmartDL and Original are 7ms and 388,620ms(normalized to 0.6 and 0.1). This shows that in this model, the convergence time of SmartDL of more than half of the mobile devices is three orders of magnitude faster than Original.

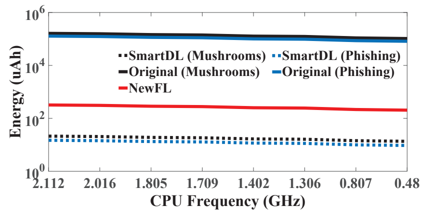
The CDF result of Figure 5(d) are similar to the previous three models, which also shows that in our simulation, compared with Original, the advantage of SmartDL is faster in training the converged model. This is because SmartDL allows the server to communicate with the participated mobile devices through the SUB method on a regular basis, and initiates central convergence when it receives more than half of the SUB signals from all selected staff or through TTL. SmartDL does not need to wait for all models to be updated. However, it can be seen that the effect of the tail of the convergence time is not good enough.

4.2.3 Comparison of Model Accuracy

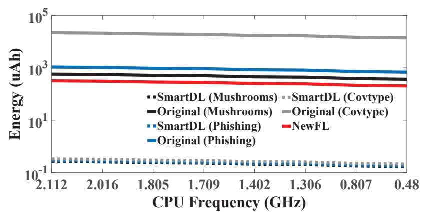
Figure 6 compares the accuracy of Tikhonov regularization model on different datasets. The result shows that the model accuracy of SmartDL is only 9% lower than that of Original in the phishing dataset. In these datasets, the housing dataset shows the largest reduction in the model accuracy of SmartDL compared with that of Original, which decreases by 12%. For the remaining datasets, the accuracy of SmartDL is almost the same as that of Original, which are all about 3%. The results show that, while guaranteeing the accuracy of the model, SmartDL effectively improves the learning speed for on-device federated learning, and it also improves the energy efficiency of federated learning on the devices from the subsequent results.



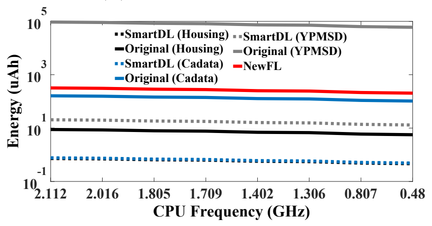
(a) Personalized PageRank.



(b) K-Nearest Neighbors.



(c) Multinomial Naïve Bayes.



(d) Tikhonov Regularization.

Fig. 7: Energy consumption during the training process with different schemes in various application scenarios.

4.2.4 Comparison of Energy Consumption

As mentioned in Section 3, SmartDL allows adaptive power control in the training algorithm rather than having a less training time. Here, we provide an energy consumption analysis based on our measured data under different CPU frequencies on the Huawei Honor 8 Lite shown in Figure 7. A common theme behind the power saving is that, no matter which baselines, the total energy consumed gradually decreases with the CPU frequency.

Figure 7(a) shows the results of the energy consumption for training of Personalized PageRank model on the movielens and jester datasets. In the movielens dataset, it can be seen that SmartDL can save 253.2uAh of energy and 3,687.1uAh of energy, as compared to NewFL and Original, respectively. In the jester dataset, SmartDL can both save about 300uAh compared to NewFL and Original.

Figure 7(b) shows the results of the energy consumption for training of K-Nearest Neighbors model on the mushrooms and phishing datasets. It can be seen that SmartDL can consume an order of magnitude less energy than NewFL, saving about 250uAh of energy. Compared to Original, SmartDL achieves energy saving in the amount of approximately 110,000uAh.

Figure 7(c) shows the results of the energy consumption for training of Multinomial Naïve Bayes model on mushrooms, phishing, and covtype datasets. Compared to NewFL, In the three datasets, SmartDL can both consume two to three orders of magnitude less energy than NewFL, saving about 263uAh of energy. In the mushrooms and phishing datasets, SmartDL consumes three orders of magnitude less energy than Original. In the the covtype dataset,

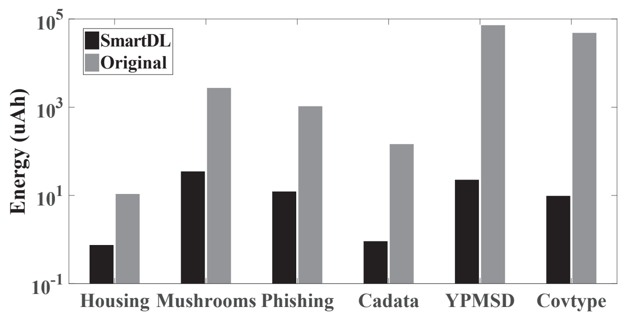


Fig. 8: Energy consumption of SmartDL and Original on Tikhonov Regularization Model with different datasets.

SmartDL consumes four orders of magnitude less energy than Original, saving about 17,908uAh of energy. Because the cardinality of the covtype dataset is much larger than the mushrooms and phishing datasets, the training time and power consumption required for a whole retraining increase accordingly.

Figure 7(d) shows the results of the energy consumption for training of Tikhonov regularization model on the housing, cadata, and YearPrediction-MSD datasets. In the housing dataset, SmartDL saves only 6.7uAh of energy compared to Original. This is because the housing dataset size is too small, so it consumes less energy for retraining. In the YearPredictionMSD dataset, SmartDL saves 77,497.6uAh of energy compared to Original.

Figure 8 compares the energy consumption of SmartDL and Original on the Tikhonov regularization model for six different datasets: housing, mushrooms, phishing, cadata, YearPreditionMSD and covtype. It can be seen that no matter what kind of dataset, SmartDL consumes at least more than one order of magnitude less energy compared to Original. Some datasets can even save three orders of magnitude of energy.

In short, compared to Original and NewFL, SmartDL can save up to 81.7% and 80.6% of energy cost on average, respectively. With the smallest dataset housing in the Tikhonov regularization model, compared with Original, SmartDL still saves 75.6% of energy. In addition, due to the different size of datasets in each model, the behave of SmartDL can change accordingly when using different datasets.

4.2.5 Comparison of Privacy Protection

Currently, there is no commonly used methodology to quantify the privacy of mobile-based deep learning [4, 42]. In order to evaluate the effectiveness of privacy protection, we propose a new metric. Specifically, in this case, we measure the privacy through calculating the proportion of the new data objects in the local training dataset. This is for the reason that, according to our discussion in Section 2 that when we need to delete some user data, federated learning can still reveal private user information based on the previous model

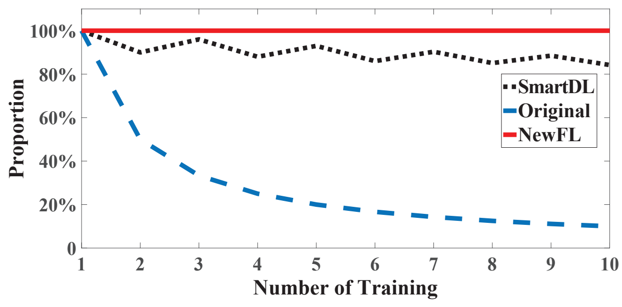


Fig. 9: Privacy under three different baselines.

clustering. In other words, when we delete old data, data privacy leak may occur. Therefore, in order to measure the privacy situation, we add 10 new data objects in each round of training, and observe the proportion of these 10 new data objects to the overall training data objects in each round. The higher the proportion, the better the privacy protection effect.

It can be seen from Figure 9 that, for NewFL, its effect is the best, it only trains new data, so its proportion is always 100%. For Original, because it needs to train all the data (10 new data and the previous old data), as the number of training increases, its proportion value continues to decrease. For SmartDL, there is a phenomenon of jitter, this is because that SmartDL includes decremental learning and incremental learning. In real life, we usually delete old data, so SmartDL pays less and less attention to old data. In addition, new data always overwrites old data. However, when we need to delete new data, SmartDL can also delete these data in a specific training round.

5 Related Work

Our work is closely related to the following three major research topics, *decremental learning*, *distributed learning* and *federated learning*.

5.1 Decremental Learning

Decremental learning is similar to the corresponding approaches used for incremental data processing [43–46] for the reason that they are both constantly updated during the training process. Ewen et al. [43] propose an approach to integrate parallel dataflows with incremental iterations. Moreover, they also provide an extension to the programming model for this workset iteration, so this improved dataflow system not only maintains a transparent and unified dataflow abstraction, but also has strong competitiveness with specialized systems. Schelter et al. [44] propose an optimistic recovery mechanism using algorithmic compensations, which takes advantage of the robustness and self-correcting nature of a large class of fix-point algorithms used in machine learning and data mining. This optimistic recovery mechanism achieves optimal failure-free performance while ensuring the overhead required for fault tolerance. McSherry et al. [45] propose a new model called differential computation, which extends the traditional incremental calculation to allow arbitrary nested iteration, and explain how to effectively implemented differential calculation in the context of a declarative data-parallel dataflow language by referring to a publicly available prototype system called Naiad. Previous studies all require a merge operation to integrate the updated data into the global results, but the difference of decreasing learning is that it also requires an inverse operation to delete the old data, but many existing studies do not cover this method.

5.2 Distributed Learning

Distributed learning is becoming more and more popular. It uses data from different places to effectively train different neural network models, which has attracted a lot of attention. Previous studies [47–57] have improved the performance of distributed learning systems from different perspectives. In order to maximize the overall performance of tasks in the cluster, Zhang et al. [47] design a scheduling algorithm to approximate the training performance of deep learning jobs. Li et al. introduce a method to manage asynchronous data communication between different working nodes with a parameter server framework that uses distributed learning, and this framework supports elastic scalability, a flexible consistency model and continuous fault tolerance. So et al. [49] propose a method to ensure the safety of the training process, which can efficiently parallelize the distributed training process and keep the training information (such as training data and model) private. Bao et al [50] design a deep learning-driven machine learning cluster scheduler, which maximizes performance and minimizes interference by placing different jobs in corresponding different machines. Jiang et al. [51] use the sketch-based method to compress the gradient values in order to accelerate the distributed learning process. Kraska et al. [52] design a novel system that can bridge the gap between end-users and distributed learning system. They not only provide a simple declarative way to specify machine learning tasks, but also provide a novel optimizer to select the corresponding learning algorithms. Mai et al. [53] design a host-based communication layer to improve the network performance in a distributed learning system. This system uses two main techniques including traffic reduction and traffic management. Moreover, Sun et al. [54] study distributed learning as a multi-agent system problem. Although these methods can effectively improve the performance of distributed learning systems, they cannot be directly used to federated learning systems based on mobile devices. The placement of training data and the execution of the training process are usually in the central servers in a distributed system. But in the federated learning system, this process is mainly carried out on mobile devices. Therefore, compared with servers located in the data center, mobile devices have higher limitations in terms of computing capacity and battery lifetime.

5.3 Federated Learning

Federated learning is proposed to enable multiple mobile devices to collaboratively train a shared deep learning model while ensuring the privacy of data [1, 15, 58–65]. Lalitha et al. [1] design a distributed learning algorithm for training models on the user networks to achieve complete decentralization. Bonawitz et al. [15] build a production system based on Tensorflow for federated learning in the field of mobile devices. Bonawitz et al. [61] also design a novel, high communication efficiency, failure-robust protocol for the safe aggregation of high-dimensional data. In order to reduce the cost of uplink communication, Konecny et al. [58] propose two methods (structured updates and ketched updates) to improve the communication efficiency in the federated learning system. In order to solve the problems of fault tolerance, high communication cost and stragglers in distributed multi-task learning, Smith et al. [59] proposed a system-aware optimization method to achieve better performance. McMahan et al. [60] design a practical method for the federated learning of deep networks based on iterative model averaging. Sprague et al. [62] propose a new algorithm, asynchronous federated learning, and study its convergence speed of training on multiple edge devices, and compare this asynchronous method with the previous synchronous method. Wang et al. [64] first theoretically explain the convergence bound of distributed gradient descent, and in order to minimize the loss function under a given resource budget, they also propose a control algorithm to determines the trade-off between local update and global parameter aggregation. Wu et al. [65] adopt a data-driven approach to introduce the opportunities and design challenges faced by Facebook in order to enable machine learning inference locally on smartphones and other edge platforms. Previous research mainly focus on reducing the communication overhead during the training process, and analyze the model convergence from a theoretical perspective. However, the problem of effectively reducing the energy consumption while ensuring the model accuracy has not been fully studied, which is essential for battery-powered mobile devices.

6 Conclusion

This paper proposes SmartDL, an energy efficient learning framework that effectively reduces the energy footprint in a Federated Learning system with a decremental learning design. SmartDL improves the energy efficiency of the training process with a hierarchical design including two layers. The first layer selects a subset of workers with sufficient capacity in order to maximize the rewards, i.e., energy saving potentials. The second layer is made up of a specified decremental learning algorithm that actively provides a decremental and incremental update functions. At the same time, it adaptively tunes the DVFS of local mobile device. SmartDL is prototyped in containerized services with modern smartphone profiles and evaluated with different learning benchmarks with real-world traces. The evaluation result shows that SmartDL achieves 75.6%–82.4% energy saving in different datasets on different models. At the same time, it achieves a speed up of 2-4 orders of magnitude comparing with the baseline. In the future work, we will evaluate SmartDL with more applications and models on more smartphones at scaling-out.

7 Acknowledgements

This work is supported by National Key R&D Program of China No. 2018YFB1404303 and ICT Grant CARCHB202017. We also acknowledge the editorial committee's support and all anonymous reviewers for their insightful comments and suggestions, which improved the content and presentation of this paper.

Conflict of interest

We wish to submit this original research article entitled "SmartDL: Energy-Aware Decremental Learning in a Mobile-based Federation for Geo-spatial System" for the consideration by *Neural Computing and Applications.*

We confirm that this work is an original journal version paper, extended from our prior position work "Exploring federated learning on battery-powered devices." in the ACM TUR-C workshop 2019, and an online report "DEAL: Decremental Energy-Aware Learning in a Federated System" in arxiv.org. Compared to previous copies, we have a completely different work on studying federated learning for geo-spatial systems. Authors have contributed on writing/proofreading this draft. We have no conflict of interest regarding to this original manuscript.

References

1. A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Third workshop on Bayesian Deep Learning (NeurIPS)*, 2018.
2. B. Nemade, "Automatic traffic surveillance using video tracking," *Procedia Computer Science*, vol. 79, pp. 402–409, 2016.
3. (2020) Big data market worth $229.4 billion by 2025. [Online]. Available: https://www.marketsandmarkets.com/PressReleases/big-data.asp
4. T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.
5. B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," *Google Research Blog*, vol. 3, 2017.
6. H. Peng, G. Liu, S. Huang, W. Yuan, and Z. Lu, "Segmentation with selectively propagated constraints," in *International Conference on Neural Information Processing*, 2016.

7. Z. Lu, Z. Fu, T. Xiang, P. Han, L. Wang, and X. Gao, "Learning from weak and noisy labels for semantic segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 3, pp. 486–500, 2016.
8. Y. Niu, Z. Lu, S. Huang, P. Han, and J.-R. Wen, "Weakly supervised matrix factorization for noisily tagged image parsing," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
9. Z. Lu, P. Han, L. Wang, and J. Wen, "Semantic sparse recoding of visual content for image applications," *IEEE Transactions on Image Processing*, vol. 24, no. 1, pp. 176–188, 2014.
10. J. Zhou, Z. Xu, W. Zheng, and Y. Wang, "Capman: Cooling and active power management in big. little battery supported devices," EasyChair, Tech. Rep., 2020.
11. (2015) Dataset. [Online]. Available: https://data.world/crowdflower/ecommerce-search-relevance
12. (2020) Gdpr.eu. recital 65: Right of rectification and erasure. [Online]. Available: https://gdpr.eu/recital-65-right-of-rectification-and-erasure
13. S. Bag, S. K. Kumar, and M. K. Tiwari, "An efficient recommendation generation using relevant jaccard similarity," *Information Sciences*, vol. 483, pp. 53–64, 2019.
14. Z. Xu, L. Li, and W. Zou, "Exploring federated learning on battery-powered devices," in *Proceedings of the ACM Turing Celebration Conference-China*, 2019, pp. 1–6.
15. K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.
16. S. Schelter, "“amnesia”-a selection of machine learning models that can forget user data very fast," *suicide*, vol. 8364, no. 44035, p. 46992, 2020.
17. Y. Zhan, P. Li, Z. Qu, D. Zeng, and S. Guo, "A learning-based incentive mechanism for federated learning," *IEEE Internet of Things Journal*, 2020.
18. G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," *Advances in neural information processing systems*, vol. 13, pp. 409–415, 2000.
19. A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app? fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*, 2012, pp. 29–42.
20. (2020) Appendix. [Online]. Available: https://github.com/good-ncu/Appendix
21. F. Li, J. Liu, and B. Ji, "Combinatorial sleeping bandits with fairness constraints," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 3, pp. 1799–1813, 2019.
22. J. Gittins, K. Glazebrook, and R. Weber, *Multi-armed bandit allocation indices.* John Wiley & Sons, 2011.
23. Y. Hou, P. Zhou, J. Xu, and D. O. Wu, "Course recommendation of mooc with big data support: A contextual online learning approach," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).* IEEE, 2018, pp. 106–111.
24. M. Chrobak and J. Noga, "Lru is better than fifo," *Algorithmica*, vol. 23, no. 2, pp. 180–185, 1999.
25. S. Wang, R. Yang, X. Xiao, Z. Wei, and Y. Yang, "Fora: simple and effective approximate single-source personalized pagerank," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 505–514.
26. P. Han, S. Shang, A. Sun, P. Zhao, and X. Zhang, "Point-of-interest recommendation with global and local context," *IEEE Transactions on Knowledge and Data Engineering*, vol. PP, no. 99, pp. 1–1, 2021.
27. P. Han, Z. Li, Y. Liu, P. Zhao, and S. Shang, "Contextualized point-of-interest recommendation," in *Twenty-Ninth International Joint Conference on Artificial Intelligence and Seventeenth Pacific Rim International Conference on Artificial Intelligence IJCAI-PRICAI-20*, 2020.
28. Y. Zhang, Y. Liu, P. Han, C. Miao, and H. Tang, "Learning personalized itemset mapping for cross-domain recommendation," in *Twenty-Ninth International Joint Conference on Artificial Intelligence and Seventeenth Pacific Rim International Conference on Artificial Intelligence IJCAI-PRICAI-20*, 2020.

29. P. Han, S. Shang, A. Sun, P. Zhao, and P. Kalnis, “Auc-mf: Point of interest recommendation with auc maximization,” in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019.
30. S. Schelter, C. Boden, and V. Markl, “Scalable similarity-based neighborhood methods with mapreduce,” in *Proceedings of the sixth ACM conference on Recommender systems*, 2012, pp. 163–170.
31. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 285–295.
32. C. Groetsch, “The theory of tikhonov regularization for fredholm equations,” *104p, Boston Pitman Publication*, 1984.
33. G. H. Golub and C. F. Van Loan, *Matrix computations.* JHU press, 2012, vol. 3.
34. L. E. Peterson, “K-nearest neighbor,” *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.
35. A. Gionis, P. Indyk, R. Motwani *et al.*, “Similarity search in high dimensions via hashing,” in *Vldb*, vol. 99, no. 6, 1999, pp. 518–529.
36. A. C. Nicholson and A. Gibson, “Deeplearning4j: Open-source, distributed deep learning for the jvm,” *Deeplearning4j. org*, 2017.
37. (2020) Monsoon power monitor. [Online]. Available: http://www.msoon.com/LabEquipment/PowerMonitor/
38. (2018) Personalized pagerank datasets. [Online]. Available: http://konect.cc/networks/
39. (2011) Classification and regression datasets. [Online]. Available: https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/
40. C. J. (2018) Federated learning - proandroiddev. [Online]. Available: https://proandroiddev.com/federated-learning-e79e054c33ef
41. C. Dinh *et al.*, “Federated learning over wireless networks: Convergence analysis and resource allocation,” *IEEE/ACM Transactions on Networking*, 2021.
42. V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, “A survey on security and privacy of federated learning,” *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2020.
43. S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl, “Spinning fast iterative data flows,” *arXiv preprint arXiv:1208.0088*, 2012.
44. S. Schelter, S. Ewen, K. Tzoumas, and V. Markl, “” all roads lead to rome” optimistic recovery for distributed iterative data processing,” in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 1919–1928.
45. F. McSherry, D. G. Murray, R. Isaacs, and M. Isard, “Differential dataflow.” in *CIDR*, 2013.
46. F. McSherry, A. Lattuada, and M. Schwarzkopf, “K-pg: Shared state in differential dataflows,” 2018.
47. H. Zhang, L. Stafman, A. Or, and M. J. Freedman, “Slaq: quality-driven scheduling for distributed machine learning,” in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 390–404.
48. M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server,” in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014, pp. 583–598.
49. J. So, B. Guler, A. S. Avestimehr, and P. Mohassel, “Codedprivateml: A fast and privacy-preserving framework for distributed machine learning,” *arXiv preprint arXiv:1902.00641*, 2019.
50. Y. Bao, Y. Peng, and C. Wu, “Deep learning-based job placement in distributed machine learning clusters,” in *IEEE INFOCOM 2019-IEEE conference on computer communications.* IEEE, 2019, pp. 505–513.
51. J. Jiang, F. Fu, T. Yang, and B. Cui, “Sketchml: Accelerating distributed machine learning with data sketches,” in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1269–1284.
52. T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, “Mlbase: A distributed machine-learning system.” in *Cidr*, vol. 1, 2013, pp. 2–1.

53. L. Mai, C. Hong, and P. Costa, “Optimizing network performance in distributed machine learning,” in *7th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.
54. S. Sun, W. Chen, J. Bian, X. Liu, and T.-Y. Liu, “Slim-dp: a multi-agent system for communication-efficient distributed deep learning,” in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 721–729.
55. S. Teerapittayanon, B. McDanel, and H.-T. Kung, “Distributed deep neural networks over the cloud, the edge and end devices,” in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 328–339.
56. P. Watcharapichat, V. L. Morales, R. C. Fernandez, and P. Pietzuch, “Ako: Decentralised deep learning with partial gradient exchange,” in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, 2016, pp. 84–97.
57. P. Han, P. Yang, P. Zhao, S. Shang, and P. Kalnis, “Gcn-mf: Disease-gene association identification by graph convolutional networks and matrix factorization,” in *the 25th ACM SIGKDD International Conference*, 2019.
58. J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.
59. V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, “Federated multi-task learning,” in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.
60. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.
61. K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.
62. M. R. Sprague, A. Jalalirad, M. Scavuzzo, C. Capota, M. Neun, L. Do, and M. Kopp, “Asynchronous federated learning for geospatial applications,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2018, pp. 21–28.
63. F. Mo and H. Haddadi, “Efficient and private federated learning using tee,” in *EuroSys*, 2019.
64. S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, “Adaptive federated learning in resource constrained edge computing systems,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.
65. C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia *et al.*, “Machine learning at facebook: Understanding inference at the edge,” in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 331–344.