Billion-Scale Federated Learning on Mobile Clients: A Submodel Design with Tunable Privacy

Chaoyue Niu, Fan Wu
Shanghai Jiao Tong University, China

Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu
Alibaba Group, China

Shaojie Tang
University of Texas at Dallas, USA

Guihai Chen
Shanghai Jiao Tong University, China

ABSTRACT

Federated learning was proposed with an intriguing vision of achieving collaborative machine learning among numerous clients without uploading their private data to a cloud server. However, the conventional framework requires each client to leverage the full model for learning, which can be prohibitively inefficient for large-scale learning tasks and resource-constrained mobile devices. Thus, we proposed a submodel framework, where clients download only the needed parts of the full model, namely, submodels, and then upload the submodel updates. Nevertheless, the "position" of a client's truly required submodel corresponds to its private data, while the disclosure of the true position to the cloud server during interactions inevitably breaks the tenet of federated learning. To integrate efficiency and privacy, we designed a secure federated submodel learning scheme coupled with a private set union protocol as a cornerstone. The secure scheme features the properties of randomized response, secure aggregation, and Bloom filter, and endows each client with customized plausible deniability (in terms of local differential privacy) against the position of its desired submodel, thereby protecting private data. We further instantiated the scheme with Alibaba's e-commerce recommendation, implemented a prototype system, and extensively evaluated over 30-day Taobao user data. Empirical results demonstrate the feasibility and scalability of the proposed scheme as well as its remarkable advantages over the conventional federated learning framework, from model accuracy and convergency, practical communication, computation, and storage overhead.

CCS CONCEPTS

• **Human-centered computing → Ubiquitous and mobile computing**; • **Computing methodologies → Machine learning**; • **Security and privacy → Distributed systems security**.

KEYWORDS

federated submodel learning, recommendation systems, local differential privacy, private set union, randomized response, secure aggregation, Bloom filter

ACM Reference Format:
Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. 2020. Billion-Scale Federated Learning on Mobile Clients: A Submodel Design with Tunable Privacy. In *MobiCom 2020 (MobiCom '20), September 21–25, 2020, London, United Kingdom.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372224.3419188

1 INTRODUCTION

1.1 Motivating Industrial Scenario

The industrial scenario in Alibaba that drives *federated submodel learning (FSL)* is the desire to provide accurate, customized, and real-time e-commerce recommendations for billion-scale mobile clients while keeping user data on local devices.

Currently, Alibaba's recommendation systems are cloud based (e.g., two-stage models with a matching stage [32, 49] for retrieving candidates and a ranking stage [14, 55, 56] for generating final recommendations, or unified models [57, 58]) and require the server cluster to collect, store, and process massive amounts of user data. Typical data fields involved in recommendations include user profile (e.g., user ID, gender, and age), user behavior (e.g., the list of visited goods IDs and relevant information, such as category IDs and shop IDs), and context (e.g., time, page number, and display position). More or less, these data fields are sensitive, and some clients who value privacy highly may refuse to share their data. In addition, according to the General Data Protection Regulation (GDPR), which took effect on May 25, 2018, any institution or company is prohibited from uploading user data (e.g., collected by apps on local devices) without the consent of European Union users [20]. Under such circumstances, refining the recommendation models and further providing accurate recommendations become practical demands.

Federated learning (FL), which decouples the ability to do machine learning from the need to upload and store data in the cloud, is a potential solution. However, the original framework of FL, proposed by Google in [34], requires each client to download the full model for training and then to upload the update of the full model, which is impractical for complex learning tasks and resource-constrained clients. Specifically, deep learning with a huge and sparse input space (e.g., e-commerce goods IDs, natural language texts, and locations) requires an embedding layer to transform inputs into a lower-dimensional space where similar inputs are close [11, 36]. In addition, the full embedding matrix tends to occupy a large proportion of the whole model parameters (e.g., 98.22% in our evaluated recommendation model and more than two-thirds in the language model of Google's Android keyboard, called Gboard [24]). Furthermore, as the largest online consumer-to-consumer platform in China, Taobao (owned by Alibaba) has roughly two billion goods

MobiCom '20, September 21–25, 2020, London, United Kingdom
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7085-1/20/09…$15.00
https://doi.org/10.1145/3372224.3419188

in total [49], which is far larger than the 10,000-word vocabulary in Gboard's natural language scenario [13, 24, 45]. This implies that the full embedding matrix of goods IDs in Taobao's deployed recommendation models roughly has two billion rows and occupies 134GB space when the embedding dimension is 18, and each element adopts 32-bit representation. If each client directly uses the full model for learning, it inevitably incurs unacceptable and unaffordable overhead for one billion Taobao users with smart devices. To improve efficiency, we observe that the input data of a certain client normally involve a small subspace of the full feature space. Thus, the client tends to need a tailored model rather than the full model, especially in the practical applications that require personalization, such as recommendation systems and multi-task learning. For example, if a Taobao user's historical data contain 300 goods, he/she requires only the corresponding 300 rows, rather than the entire two billion rows, of the full embedding matrix. Based on this key observation, we propose a general and scalable FSL framework.

1.2 Federated Submodel Learning Framework

In each communication round of FSL, a cloud server first selects some eligible clients to participate. Then, each chosen client downloads part of the global model as required, namely, a submodel, from the cloud server. For example, in the e-commerce recommendation, a client's submodel mainly contains the embedding vectors for the goods IDs in its historical data, as well as the parameters of the other network layers. Then, the client trains the submodel over its local data and uploads the submodel update. The cloud server aggregates the submodel updates from live clients and forms a consensus update to the global model. The process above is iterated continuously in practice so that the global model on the cloud server and the submodels on the clients are kept up-to-date.

If each client leverages the full model rather than its required submodel for learning, FSL will degenerate to conventional FL. FSL further decouples the ability to accomplish FL from the need to use the prohibitively large full model, dramatically improving efficiency. For example, in our evaluation, the size of a client's desired submodel is only 1.99% of the full model's size. The generalization of FSL also implies that almost all existing methods to improve the efficiency and scalability of FL can still apply to FSL. For example, the model compression algorithms not only can compress the full model (update) but also can compress the submodel (update) to reduce overhead. Therefore, FSL is more practical for distributing industrial-strength learning tasks on ubiquitous mobile devices.

1.3 Newly Introduced Privacy Risks

Just as every coin has two sides, FSL not only provides efficiency but also introduces two extra privacy risks.

First, compared with using the public full model in FL, the download of a submodel and the upload of the submodel update would require each client to provide an index set as auxiliary information, specifying the "position" of its submodel. However, the real index set normally corresponds to the client's private data. For example, to specify the required rows of the embedding matrix in the e-commerce scenario, a client mainly needs to provide the goods IDs in its user data as the real index set. Similarly, in the natural language scenario, a client's real index set to locate the desired word embedding vectors is actually the vocabulary extracted from the client's typed texts. Thus, the disclosure of a client's real index set to the cloud server can still be regarded as the leakage of the client's private data, breaking the tenet of FL.

Second, compared with the aligned full model in FL, each client chosen in one round of FSL submits only the update of its customized submodel, which tends to be highly differentiated from the other chosen clients' submodels. As a result, the aggregation of updates w.r.t. a certain index can come from a unique client out of the chosen ones (e.g., with probability 86.7% for 100 clients randomly chosen in our Taobao dataset). This implies that from the aggregate update, the cloud server not only can ascertain that the client has the index but also can learn its detailed update. Besides the real index revealing a client's private data, the client's individual update can still memorize or even allow reconstruction of its private data, namely, the model inversion attack [22, 59] on the individual client. Further, as the goods IDs of different Taobao users are more heterogeneous than the vocabularies of different Gboard users, the misalignment of desired submodels and the privacy risk in e-commerce are more severe than those in natural language.

1.4 Fundamental Problems and Challenges

To mitigate the privacy risks above, we need to jointly solve two fundamental problems abstracted from the download and upload phases of FSL: (1) How a client can download a row of a matrix, which represents the global/full model and is maintained by an untrusted cloud server, without revealing which row or the row index to the cloud server; and (2) how a client can modify a row of the matrix, still without revealing which row was modified and the altered content. We analyze two problems in detail.

We start with the first problem. One naïve method is that the client downloads the full matrix, as in conventional FL, and then extracts the required row locally. This method perfectly hides the fetched row index, but incurs immense communication overhead. To avoid downloading the full matrix, *private information retrieval (PIR)* [4, 12, 44] can be applied, which exactly matches the problem settings, including the read-only mode and the concealment of the retrieved elements. Therefore, if we consider the first problem independently, PIR may be a good choice.

We next deconstruct the second problem. For a concrete row of the full matrix, if clients modify the row one by one, the cloud server definitely knows the clients who modified this row and their detailed modifications. Thus, one feasible way is to first securely aggregate (i.e., add up) all the modifications without revealing any individual modification and then apply the aggregate modification (i.e., the sum of the modifications) to the row of the full matrix once. Such a primitive of oblivious addition can be provided by many cryptographic schemes (e.g., the protocol specific to the FL setting in [8] and additively homomorphic encryption [9, 43]). With the secure aggregation guarantee, if more than one client participates in aggregation and at least one of their modifications is nonzero, then the cloud server cannot know any individual modification and thus cannot reveal which client(s) truly intend to modify this row. Further, a larger number of involved clients imply better privacy. One extreme case is in conventional FL with secure aggregation, also called *secure federated learning (SFL)*, which harshly lets all chosen clients in one round be involved for each row of the full matrix, no matter whether they truly intend to modify this row or not. Thus, SFL offers the best privacy but the worst efficiency. Another extreme case is directly combining FSL with secure aggregation, which simply lets those clients who really intend to modify a row be involved. Thus, each chosen client needs to be involved only for those rows that it truly intends to modify, implying the best efficiency. However, different clients tend to modify highly differentiated or even mutually exclusive rows. For joint modification w.r.t. some row, chances are high that only one client is involved. Under this circumstance, the secure aggregation guarantee no longer works, while the client's real intention and detailed modification are disclosed. In a nutshell, trivial solutions to the second problem cannot achieve a tunable balance between privacy and efficiency.

1.5 Solution Overview and Contributions

Considering the two fundamental problems above and several practical issues together, we propose *secure federated submodel learning (SFSL)*. We first determine the scope of joint modification in each round of FSL, thereby aligning differentiated submodels. The key obstacle is the need to conceal the position of any chosen client's desired submodel (alternatively, its real index set or private data). The baseline SFL trivially uses the prohibitively large full index set (i.e., the position of the full model) for alignment. In contrast, our SFSL identifies a necessary and sufficient scope for alignment, namely, the union of the chosen clients' real index sets. The union is efficiently obtained through our proposed *private set union (PSU)* protocol without revealing any individual real index set. In addition, because the union is (far in our application scenario) smaller than the full index set, SFSL can significantly outperform SFL without sacrificing privacy. Based on the union, each chosen client generates a randomized index set to replace and protect its real index set for interactions in the download and upload phases. Specifically, the randomized index set is generated by applying randomized response twice with one memoization step between, and the parameter setting in randomized response is customized by the client. Such a design with secure aggregation allows the client to hold self-controllable deniability against whether it really intends or does not intend to download some row and to upload the modification of this row, even if the client participates in multiple rounds. The strength of deniability is rigorously quantified using *local differential privacy (LDP)*. Meanwhile, the probability of the cloud server inferring the client's real intention from the aggregate modification is also thoroughly analyzed. Furthermore, the cardinality of the randomized index set dominates the client's overhead, and the intersection of the client's real and randomized index sets controls its local training. Therefore, each client can fine-tune the tension among privacy, efficiency, and effectiveness when using SFSL.

We summarize the key contributions of this work as follows:

- To the best of our knowledge, we are the first to propose the FSL framework for industrial-strength, on-device intelligence, and further to identify and remedy privacy risks.
- The proposed secure scheme SFSL empowers each client with finely tunable deniability against its real intention of downloading the desired submodel and uploading the submodel update, thereby protecting private data.
- As a moat, we designed an efficient and scalable PSU protocol using Bloom filter, secure aggregation, and randomization, which can have independent and significant value in practice.
- We instantiated SFSL with Alibaba's e-commerce scenario, adopted the *deep interest network (DIN)* for recommendation, and implemented a prototype system. We also conducted extensive evaluations using the data of Taobao users collected over a period of 30 days. When the number of chosen clients per round is 100, the major experimental results are presented as follows: (1) Compared with conventional FL, which diverges in the end, SFSL improves the highest *area under the curve (AUC)* by 7.22%; (2) at the same security and privacy levels as SFL, SFSL reduces 80.05% of the communication overhead on both sides of the client and the cloud server, and reduces 85.02% (resp., 45.43%) and 72.51% (resp., 63.77%) of the computation (resp., memory) overhead on the sides of the client and the cloud server, respectively; and (3) for the proposed PSU protocol, the communication overhead per client is less than 1MB. The computation overhead of the client and the cloud server is each less than 40s, even if the dropout ratio of the chosen clients reaches 20%.

2 RELATED WORK

In recent few years, FL has become an active topic in academic and industrial fields. In this section, we focus on the related work about security and privacy. For other focuses, we direct interested readers to comprehensive surveys [26, 31, 54].

Bonawitz et al. [8] designed a secure aggregation protocol in both honest-but-curious and actively adversarial settings, which will be used as a building block of our SFSL. In particular, their design was specific to the FL framework, where each resource-constrained client cannot establish direct communication channels with other clients and needs to rely on an untrusted cloud server as a relay, while part of clients may drop out during the aggregation process. To bound the leakage of each individual client's training data from the model updates, several differentially private mechanisms were proposed. McMahan et al. [35] considered recurrent language models, let the trusted cloud server perturb the aggregate model update in a single round with a Gaussian mechanism, and relied on the celebrated moments account scheme [1] for multi-round privacy composition. Agarwal et al. [2] studied how to integrate *differential privacy (DP)* with model compression and proposed a Binomial mechanism for each client to perturb its quantized model update. The quantization [30, 48, 53] is to encode original float-type and continuous parameters into integer-type and discrete values with a few bits. In contrast to these defense mechanisms, Bagdasaryan et al. [6] developed a model replacement attack launched by malicious clients to backdoor the global model on the cloud server. Nasr et al. [40] considered the membership inference attack in pure FL without secure aggregation and leveraged a client's individual model update to uncover the membership of its training data.

Parallel to existing work in FL, where each client uses the same global model for learning, we propose a novel FSL framework for scalability. Under FSL, we focus on new privacy issues due to the dependence between the position of a client's desired submodel and its private data, as well as the misalignment of the submodel updates from heterogeneous clients in secure aggregation.

3 PRELIMINARIES

In this section, we define security requirements corresponding to the privacy risks introduced in Section 1.3. We also review a primitive building block, called randomized response.

We first introduce some frequently used notations. We use a two-dimensional matrix with m rows and d columns to represent the global/full model, denoted as $\mathbf{W}$. We also let $\mathcal{S} = \{1, 2, \ldots, m\}$ denote the full (row) index set of $\mathbf{W}$. In addition, we let C denote n clients who are selected by the cloud server to participate in any round of FSL. For a chosen client $i \in C$, we let $\mathcal{S}^{(i)} \subset \mathcal{S}$ denote its real index set and let $\mathbf{W}_{\mathcal{S}^{(i)}}$ denote its desired submodel, which imply that the user data of client i involve (intuitively, after local training may update) the rows in $\mathbf{W}$ with indices $\mathcal{S}^{(i)}$.

3.1 Security Requirements

First, for the disclosure of real index sets when clients interact with the untrusted cloud server, we consider that each client should have plausible deniability of whether a certain index is or is not in its real index set. To measure the strength of plausible deniability, we adopt LDP, which is a variant of standard DP in the local setting. Specifically, the randomization in LDP is performed by clients in a distributed manner, rather than relying on a data curator as a trusted authority to conduct centralized perturbation in DP. Thus, the privacy of an individual client's data is protected not only from external attackers but also from the untrusted data curator (e.g., the cloud server in FSL). LDP for various population statistics has received industrial deployments (e.g., by Google [19, 21], Apple [5], and Microsoft [16]) as well as extensive academic attention [18, 27, 50, 51]. We present the formal definition of LDP as follows:

Definition 3.1. A randomized mechanism M satisfies ϵ-LDP, if for any pair of inputs from a client, denoted as x and y, and for any possible output of M, denoted as z, we have $\frac{\Pr(M(x)=z)}{\Pr(M(y)=z)} \leq \exp(\epsilon)$. A smaller privacy budget ϵ offers a better LDP guarantee.

Intuitively, LDP says that the output distribution of the randomized mechanism does not change too much, given distinct inputs from the client. Thus, LDP formalizes a sort of plausible deniability: No matter what output is revealed, it is approximately equally as likely to have come from one input as any other input. When LDP applies to obscure the membership of a certain index in FSL, the inputs and outputs are boolean values, where possible inputs (resp., outputs) are two states: a certain index "in" or "not in" a client's real (resp., revealed) index set. We can check that FL provides the strongest deniability, where the level of LDP is $\epsilon = \ln(1/1) = 0$ for each client. The reason is that no matter whether an index is or is not in a client's real index set (different inputs), this index will definitely be revealed (the same output "in"). In contrast, FSL provides the weakest deniability, where the level of LDP is $\epsilon = \ln(1/0) = \infty$ for each client, because if an index is in (resp., not in) a client's real index set, this index will definitely (resp., definitely not) be revealed, namely, output "in" with probability 1 (resp., 0).

Second, for the secure aggregation of misaligned submodel updates in any round, we consider that the clients chosen in the round should be enabled to tune the privacy level, rather than trivially choosing extreme SFL or FSL with secure aggregation. We define a client-tunable privacy protection mechanism as follows:

Definition 3.2. A privacy protection mechanism for aggregating submodel updates is client tunable, if participating clients can control the probabilities of the following two events from the securely aggregated submodel update.

- *Event 1:* The cloud server ascertains that an index belongs to some client and learns its detailed update w.r.t. this index.
- *Event 2:* The cloud server ascertains that an index does not belong to some client.

If we view the disclosure of a client's real intention w.r.t. an index (i.e., the occurrence of either Event 1 or Event 2) as a breach of deniability about the index, then Definition 3.2 complements Definition 3.1 for secure aggregation of submodel updates. If we view the disclosure of a client's individual submodel update as the premise of the model inversion attack (resp., the membership inference attack) to reveal the content (resp., the membership) of the client's private data, Definition 3.2 can also be interpreted from mitigating the attack on each individual client. We first examine SFL. If at least two clients participate, the probability of Event 1 is 0, and the probability of Event 2 is still 0 for those indices in the union of the chosen clients' real index sets. For an index outside the union, the probability of Event 2 approaches 1. Although each chosen client submits a zero vector w.r.t. this index, from the aggregate zero vector, the cloud server almost ascertains that all the chosen clients do not have the index, despite of some rare cases. Regarding directly combining FSL with secure aggregation, the probability of Event 1 depends entirely on the heterogeneity of the user data on the clients, and the probability of Event 2 is the same as that in SFL.

3.2 Randomized Response

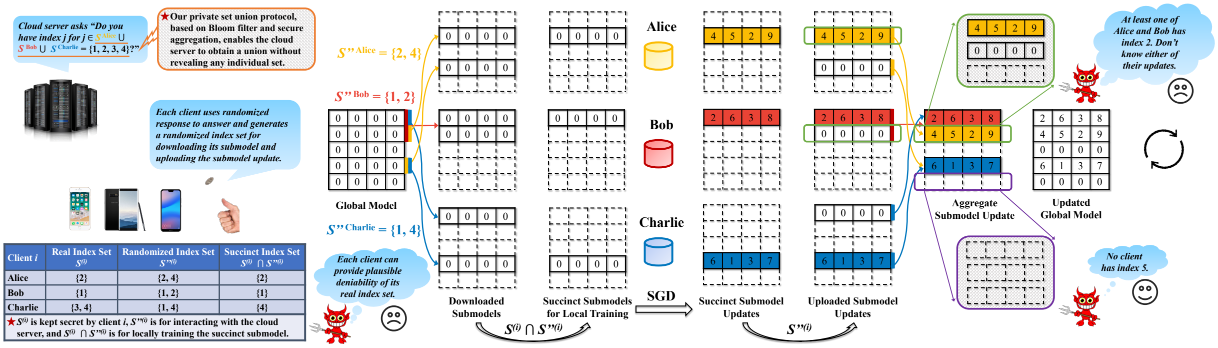
Randomized response, due to Warner in 1965 [52], is a survey technique in the social sciences for collecting statistics about illegal, embarrassing, or sensitive topics, where the respondents want to preserve the privacy of their answers. A classic example for illustration is the "Are you a member of the Communist Party?" question. For this question, each respondent flips a fair coin in secret and tells the truth if it comes up tails; otherwise, he/she flips a second coin and responds "Yes" if heads and "No" if tails. Thus, a communist (resp., non-communist) will answer "Yes" with probability 75% (resp., 25%) and "No" with probability 25% (resp., 75%).

Randomized response can provide plausible deniability for both "Yes" and "No" answers. A communist can contribute his/her response of "Yes" to the event that the first and second coin flips were both heads, which occurs with probability 25%. Meanwhile, a non-communist can also contribute his/her response of "No" to the event that the first coin was heads and the second coin was tails, which still occurs with probability 25%. Furthermore, the plausible deniability of a one-time randomized response can be rigorously quantified by LDP, particularly at the level $\epsilon = \ln(75\%/25\%) = \ln 3$, irrespective of any attacker's prior knowledge [17, 19].

4 DESIGN OF SFSL

4.1 Design Rationale

We illustrate key design principles mainly through demonstrating how to handle the two fundamental problems raised in Section 1.4 and how to resolve several practical issues.


Cloud server asks "Do you have index j for $j \in S^{\text{Alice}} \cup S^{\text{Bob}} \cup S^{\text{Charlie}} = \{1, 2, 3, 4\}$?"
★Our private set union protocol, based on Bloom filter and secure aggregation, enables the cloud server to obtain a union without revealing any individual set.
Each client uses randomized response to answer and generates a randomized index set for downloading its submodel and uploading the submodel update.
$S''^{\text{Alice}} = \{2, 4\}$
$S''^{\text{Bob}} = \{1, 2\}$
$S''^{\text{Charlie}} = \{1, 4\}$
Global Model
Each client can provide plausible deniability of its real index set.
Downloaded Submodels
Succinct Submodels for Local Training
$S^{(i)} \cap S''^{(i)}$
Alice
Bob
Charlie
SGD
Succinct Submodel Updates
Uploaded Submodel Updates
$S''^{(i)}$
Aggregate Submodel Update
Updated Global Model
At least one of Alice and Bob has index 2. Don't know either of their updates.
No client has index 5.

Client i	Real Index Set $S^{(i)}$	Randomized Index Set $S''^{(i)}$	Succinct Index Set $S^{(i)} \cap S''^{(i)}$
Alice	{2}	{2, 4}	{2}
Bob	{1}	{1, 2}	{1}
Charlie	{3, 4}	{1, 4}	{4}

★$S^{(i)}$ is kept secret by client i, $S''^{(i)}$ is for interacting with the cloud server, and $S^{(i)} \cap S''^{(i)}$ is for locally training the succinct submodel.

Figure 1: An illustration of SFSL. The gray rounded rectangle denotes secure aggregation. The randomized index sets of Alice, Bob, and Charlie are $\{2,4\}$, $\{1,2\}$, and $\{1,4\}$, respectively. For index 2, Alice submits the update $(4,5,2,9)$, Bob pretends to submit a zero vector, while Charlie does not submit. Secure aggregation ensures that the adversarial cloud server obtains only the sum of Alice's and Bob's updates $(4,5,2,9)$, but does not know either update. Thus, the cloud server can only infer that at least one of Alice and Bob has index 2. Due to plausible deniability, the cloud server also cannot ascertain the state of Charlie.

As shown in Figure 1, we handle two fundamental problems in a unified manner rather than in separate ways. During the download and upload phases, a client consistently uses a randomized index set in place of its real index set. In contrast, during the local training phase, the client leverages the intersection of its real and randomized index sets (called the client's succinct index set) to prepare the succinct submodel and the user data involved. With the blind of the randomized index set to interact with the outside world, the client holds plausible deniability of some index being in or not in its real index set. Specifically, the client generates its randomized index set locally with randomized response as follows. The sensitive question asked by the cloud server is "Do you have a certain index?". Then, the client answers "Yes" with two customized probabilities, conditional on whether the index is or is not in the client's real index set, and puts the index into the randomized index set if it receives a "Yes" answer. The two probabilities allow the client to fine-tune the balance between deniability and utility.

We further examine the feasibility of index set randomization in handling two problems. For the first problem in the download phase, if a client intends (resp., does not intend) to download a certain row, and it actually downloads (resp., does not download), it can blame its action to randomization, i.e., the occurrence of the event that the index not in (resp., in) a client's real index set returns a "Yes" (resp., "No") answer. Regarding the second problem in the upload phase, the usage of the randomized index set still empowers a client to deny its real intention of modifying or not modifying some row, even if the cloud server observes its binary action of modifying or not modifying. Additionally, for a concrete row, there are two different groups of clients involved in the joint modification: (1) One group consists of those clients who intend to modify the row and contribute nonzero modifications; and (2) the other group comprises those clients who do not intend to modify the row and pretend to modify by submitting zero modifications. With the secure aggregation guarantee, even though the adversarial cloud server observes the aggregate modification, it is hard to identify any individual modification and further to infer whether some client originally intends to perform a modification or not. The hardness is controlled by the sizes of two groups, alternatively, the probabilities of an index in and not in the real index set returning a "Yes" answer, which are fully tunable by clients as expected in Definition 3.2.

Besides two fundamental problems, there still exist two practical issues to be solved before the method of index set randomization can apply to FSL. The first issue regards whether it is practical and necessary for the cloud server to ask "Do you have a certain index?" for each index in the full index set, which is intrinsically adopted by FL. For example, the matrix, representing Taobao's full recommendation model, has billions of rows. Thus, it is impractical for a client to answer billion-scale questions and further to download and securely upload those rows with "Yes" answers. We turn to narrowing down the scope to the union of n chosen clients' real index sets, which is normally far smaller than the full index set (i.e., the union of the whole clients' real index sets). Our optimization is inspired by an observation: If a client's real index set is of size 300, and the full index set is of size 2 billion, using the probability parameters in the survey of party membership, the expected number of "Yes" answers is $300 \times 75\% + (2 \times 10^9 - 300) \times 25\% \approx 5 \times 10^8$. Such a calculation implies that the dominant "Yes" answers are those with "No" in reality but "Yes" due to randomness. Nevertheless, most of the "No"-to-"Yes" answers are useless. Specifically, for those indices that do not belong to any chosen client's real index set (e.g., index 5 in Figure 1), although part (25% in expectation) of the chosen clients upload zero vectors for randomization, the cloud server can still infer from the aggregate zero vectors that these clients do not actually have the indices. Therefore, it is unnecessary to cover any index outside the union of n chosen clients' real index sets. By using the union, we set $n = 100$ and recalculate the expected number of "Yes" answers as $300 \times 75\% + (100 \times 300 - 300) \times 25\% \approx 8 \times 10^3$. This implies that a prohibitively large number of unnecessary "No"-to-"Yes" answers are avoided, significantly improving efficiency and without degrading privacy. Now, a basic and thorny problem that accompanies is how multiple clients can obtain the union of their real index sets under the mediation of an untrusted cloud server without revealing any client's real index set, namely, the need of a PSU protocol. Considering existing schemes cannot satisfy the atypical setting of FSL and high scalability, we propose a novel design of PSU based on Bloom filter and secure aggregation.

The second issue regards the longitudinal privacy when a client is chosen to participate in multiple rounds of FSL. The initial version of randomized response provides a rigorous privacy guarantee only when an audience answers the same question once, facilitating a one-time response to "Do you have a certain index?". Thus, we need to extend the initial version to allow repeated responses from the same client to those already answered indices. Our extension leverages key principles from *randomized aggregatable privacy-preserving ordinal response (RAPPOR)* [19, 21] and plays randomized response twice with a memoization step between. In particular, the noisy answers generated by the inner (permanent) randomized response will be memoized and permanently replace the real answers in the outer (instantaneous) randomized response. This ensures that even though a client responds to the membership of a concrete index for infinite times, the client can still hold plausible deniability of its real answer, where the level of deniability is lower bounded by the privacy level of the memoized noisy answer.

4.2 Design Details

We introduce the design details of SFSL in a top-down manner. We first give an overview of its top-level architecture and then show two underlying modules, namely, index set randomization and PSU.

4.2.1 Secure Federated Submodel Learning (SFSL). Before getting into SFSL, we first review federated averaging [34], which is the default distributed optimization algorithm of FL. In each round, the cloud server restarts a collaborative training process and randomly chooses some eligible clients to locally train the latest global model. Then, the cloud server takes a weighted average of the full model updates, one chosen client's weight being proportional to the size of its training set, and adds the aggregate update to the global model.

We now present SFSL in Algorithm 1, which generalizes federated averaging to support effective and efficient FSL. SFSL also preserves desired security and privacy properties while incorporating the unstable and limited network connections of mobile devices. At the initial stage, the cloud server randomly initializes the global model (Line 1). In each communication round, the cloud server selects n clients to participate (Lines 2 and 3) and also maintains an up-to-date set of the chosen clients who are alive throughout the whole round, denoted by $\hat{C}$. A chosen client determines its real index set based on its local data, which specifies the "position" of its truly required submodel (Line 10). For example, if the goods IDs of a Taobao user include $\{1, 2, 4\}$, then he/she requires the first, second, and fourth rows of the embedding matrix for goods IDs, further implying that the real index set should contain $\{1, 2, 4\}$. Then, the cloud server launches PSU to obtain the union of the chosen clients' real index sets while keeping each individual client's real index set in secret (Lines 4 and 11). The union is delivered to live clients for them to generate randomized index sets with customized LDP guarantees against the cloud server (Line 12). Each live client will use its randomized index set, rather than its real index set, to download the submodel and to securely upload the submodel update (Lines 13 and 19). Upon receiving the randomized index set from a client, the cloud server stores it for later usage and returns the corresponding submodel and training hyperparameters to the client (Line 6).

Algorithm 1: Secure Federated Submodel Learning (SFSL)

```
   /* Cloud server's process */
1  Initialize the global model $\mathbf{W}$;
2  foreach communication round do
3      Randomly select $n$ clients, denoted as $C$;
4      Launch private set union (Algorithm 3), obtain the union of $n$
         clients' real index sets, namely, $\bigcup_{i \in C} \mathcal{S}^{(i)}$, and deliver the
         union result to the up-to-date set of the clients who are alive,
         denoted as $\hat{C} \subset C$;
5      foreach client $i \in \hat{C}$ do
6          Receive and store the randomized index set $\mathcal{S}''^{(i)}$ from
             client $i$, and return the submodel $\mathbf{W}_{\mathcal{S}''^{(i)}}$ and training
             hyperparameters to $i$;
7      foreach $j \in \bigcup_{i \in C} \mathcal{S}^{(i)}$ do
8          Determine the live clients involving index $j$, denoted as
             $\hat{C}_j = \{i | i \in \hat{C} \wedge j \in \mathcal{S}''^{(i)}\}$, let them submit materials
             for secure aggregation, and obtain the sum of (weighted)
             updates $\sum_{i \in \hat{C}_j} \Delta\mathbf{w}_j^{(i)}$ and the total count number of
             relevant samples $\sum_{i \in \hat{C}_j} v_j^{(i)}$;
9          Update the $j$-th row of the global model $\mathbf{W}$ by adding
             $\sum_{i \in \hat{C}_j} \Delta\mathbf{w}_j^{(i)} \Big/ \sum_{i \in \hat{C}_j} v_j^{(i)}$;
   /* Client $i$'s process */
10 Determine its real index set $\mathcal{S}^{(i)}$ based on local data;
11 Participate in private set union (Algorithm 3);
12 Generate a randomized index set $\mathcal{S}''^{(i)}$ (Algorithm 2);
13 Use $\mathcal{S}''^{(i)}$ to download a submodel, denoted as $\mathbf{W}_{\mathcal{S}''^{(i)}}$;
14 Depending on the succinct index set $\mathcal{S}''^{(i)} \bigcap \mathcal{S}^{(i)}$, locally extract
     the succinct submodel $\mathbf{W}_{\mathcal{S}''^{(i)} \bigcap \mathcal{S}^{(i)}}$ from $\mathbf{W}_{\mathcal{S}''^{(i)}}$ and prepare
     involved data as the succinct training set;
15 Train $\mathbf{W}_{\mathcal{S}''^{(i)} \bigcap \mathcal{S}^{(i)}}$ using the hyperparameters and obtain the
     update of the succinct submodel $\Delta\mathbf{W}_{\mathcal{S}''^{(i)} \bigcap \mathcal{S}^{(i)}}$;
16 Initialize the submodel update to be uploaded with $\mathcal{S}''^{(i)}$, denoted
     as $\Delta\mathbf{W}_{\mathcal{S}''^{(i)}}$, all to 0, and add $\Delta\mathbf{W}_{\mathcal{S}''^{(i)} \bigcap \mathcal{S}^{(i)}}$;
17 Count the number of training samples involving each index
     $j \in \mathcal{S}''^{(i)}$ and store the results in the vector $\mathbf{v}_{\mathcal{S}''^{(i)}}$;
18 Weight $\Delta\mathbf{W}_{\mathcal{S}''^{(i)}}$ in advance by multiplying each row with the
     corresponding count number in $\mathbf{v}_{\mathcal{S}''^{(i)}}$;
19 Upload materials for securely aggregating $\Delta\mathbf{W}_{\mathcal{S}''^{(i)}}, \mathbf{v}_{\mathcal{S}''^{(i)}}$.
```

Depending on the intersection of the real and randomized index sets (i.e., the succinct index set), the client extracts a succinct submodel and prepares involved data as the succinct training set (Line 14). For example, if a Taobao user's real index set is $\{1, 2, 4\}$, and his/her randomized index set is $\{2, 4, 6, 9\}$, he/she receives a submodel with the row indices $\{2, 4, 6, 9\}$ from the cloud server, but just needs to train the succinct submodel with the row indices $\{2, 4\}$ over his/her local data involving the goods IDs $\{2, 4\}$. After training under the preset hyperparameters, the client obtains the update of the succinct submodel (Line 15). Then, it prepares the submodel update to be uploaded with the randomized index set by adding the update of the succinct submodel to the rows with the succinct indices and padding zero vectors to the other rows (Line 16). To help the cloud server average multiple submodel updates according to the size of relevant local training data, each chosen client also needs to count the number of its training samples involving every index in the randomized index set (Line 17). In particular, the numbers of samples involving the indices outside the succinct index set are all zeros. Furthermore, each client weights the submodel update to be uploaded in advance by multiplying each row with the corresponding count number, namely, the weight (Line 18).

The weighted submodel updates and the count vectors from live clients are securely aggregated side by side under the coordination of the cloud server (Lines 7–9 and 19). Specifically, the cloud server guides the secure aggregation by enumerating every index in the union of real index sets as follows. It first determines the set of live clients whose randomized index sets contain this index and then lets these clients submit the materials for securely adding up the weighted updates and the count numbers w.r.t. the index (Line 8). The cloud server finally applies an aggregate update to the global model in this row by adding the quotient between the sum of the weighted updates and the total count number, namely, the weighted average (Line 9). To reduce the interactions between the cloud server and any client, they can package all the materials supporting secure aggregation, rather than exchanging for one index each time (Lines 7–9). In particular, the cloud server executes Lines 7 and 8 for each live client $i \in \hat{C}$ in parallel and then executes Line 9 for each index in the union $j \in \bigcup_{i \in C} \mathcal{S}^{(i)}$.

4.2.2 Index Set Randomization. We next show how a client can generate a randomized index set in each participating round.

We start with the basic design. The sensitive question, asked by the cloud server, is "Do you have a certain index?". All the clients chosen in one round of FSL make up the population to be surveyed, while the union of their real index sets works as the necessary and sufficient scope of the questionnaire. Given the questionnaire, client i leverages randomized response to answer "Yes" or "No" and puts the indices with "Yes" answers into its randomized index set. To fine-tune the tension among effectiveness, efficiency, and privacy, client i basically adjusts two probability parameters $p_1^{(i)}, p_2^{(i)}$ in randomized response. In particular, $p_1^{(i)}$ denotes the probability that an index in client i's real index set will return a "Yes" answer and controls the factual size of client i's data contributed to FSL. A larger $p_1^{(i)}$ generally implies better effectiveness from convergence rate. In addition, $p_2^{(i)}$ denotes the probability that an index outside client i's real index set will return a "Yes" answer and determines the number of redundant rows to be downloaded and the number of padded zero vectors to be uploaded through secure aggregation. Hence, given a fixed $p_1^{(i)}$, a smaller $p_2^{(i)}$ indicates better efficiency. Furthermore, $p_1^{(i)}, p_2^{(i)}$ jointly adjust the level of LDP, where a pair of closer $p_1^{(i)}, p_2^{(i)}$ provide better LDP. We examine three typical instances: (1) The party membership survey takes $p_1^{(i)} = 75\%, p_2^{(i)} = 25\%$ for each respondent; (2) FL essentially uses the full index set as the scope of the cloud server's questionnaire, takes $p_1^{(i)} = p_2^{(i)} = 1$ for each client, and offers the best LDP and effectiveness but the worst efficiency; and (3) FSL adopts $p_1^{(i)} = 1, p_2^{(i)} = 0$ for each client and provides the best effectiveness and efficiency but the worst LDP.

We further extend the basic design to allow multi-round participation of client i and its repeated responses to any index. As shown in Algorithm 2, we let client i maintain two index sets with "Yes" and "No" answers in the permanent randomized response, respectively (Input). The permanent randomized response is parameterized by $p_1^{(i)}, p_2^{(i)}$ as illustrated above. Given that a client can be grouped with distinct clients in different rounds while the union of the chosen clients' real index sets varies from one round to another, the client needs to handle new indices. For a certain round, as any new index comes (Line 2), client i first generates a permanent noisy answer for it, conditional on whether the new index is in or is not in the real index set (Lines 3–6). Then, client i updates two memoized index sets (Lines 7–10). Based on the memoized noisy answers to the union of real index sets, client i obtains its final randomized index set by performing an instantaneous randomized response over the union (Lines 11–16). The instantaneous randomized response is parameterized with another two probabilities $p_3^{(i)}, p_4^{(i)}$, similar to $p_1^{(i)}, p_2^{(i)}$ in the permanent randomized response. Now, these four probability parameters jointly support tuning the tension among privacy, effectiveness, and efficiency. In particular, $p_5^{(i)} = p_1^{(i)}(p_3^{(i)} - p_4^{(i)}) + p_4^{(i)}$ (resp., $p_6^{(i)} = p_2^{(i)}(p_3^{(i)} - p_4^{(i)}) + p_4^{(i)}$), denoting the probability that an index in (resp., not in) client i's real index set finally returns a "Yes" answer, now plays the same role as $p_1^{(i)}$ (resp., $p_2^{(i)}$) in the basic design. Detailed derivations of $p_5^{(i)}, p_6^{(i)}$ can be found in the complete manuscript [42].

Algorithm 2: Client i's Index Set Randomization

```
Input: Client $i$'s real index set $\mathcal{S}^{(i)}$, Union of the chosen clients'
       real index sets $\bigcup_{i \in C} \mathcal{S}^{(i)}$, Client $i$'s memoized index set
       $\mathcal{Y}^{(i)}$ (resp., $\mathcal{N}^{(i)}$) initialized to $\emptyset$ at very beginning with
       permanent "Yes" (resp., "No") answers to the question "Do
       you have a certain index?", Client $i$'s customized probability
       parameters $0 \le p_1^{(i)}, p_2^{(i)}, p_3^{(i)}, p_4^{(i)} \le 1$.
Output: Client $i$'s doubly randomized index set $\mathcal{S}''^{(i)}$
1  $\mathcal{S}'^{(i)} = \emptyset, \mathcal{S}''^{(i)} = \emptyset$;
   // Permanent randomized response
2  foreach $j \in \bigcup_{i \in C} \mathcal{S}^{(i)} \wedge j \notin \mathcal{Y}^{(i)} \bigcup \mathcal{N}^{(i)}$ do
3      if $j \in \mathcal{S}^{(i)}$ then
4          Add $j$ to $\mathcal{S}'^{(i)}$ with probability $p_1^{(i)}$;
5      else
6          Add $j$ to $\mathcal{S}'^{(i)}$ with probability $p_2^{(i)}$;
       // Memoization of permanent answers
7      if $j \in \mathcal{S}'^{(i)}$ then
8          $\mathcal{Y}^{(i)} = \mathcal{Y}^{(i)} \bigcup j$;
9      else
10         $\mathcal{N}^{(i)} = \mathcal{N}^{(i)} \bigcup j$;
   // Instantaneous randomized response
11 foreach $j \in \bigcup_{i \in C} \mathcal{S}^{(i)}$ do
12     if $j \in \mathcal{Y}^{(i)}$ then
13         Add $j$ to $\mathcal{S}''^{(i)}$ with probability $p_3^{(i)}$;
14     else
15         Add $j$ to $\mathcal{S}''^{(i)}$ with probability $p_4^{(i)}$;
16 return $\mathcal{S}''^{(i)}$
```

4.2.3 Private Set Union (PSU). We introduce the last PSU module. We first briefly review existing PSU protocols and illustrate their practical infeasibility in FSL. We then present our new design.

Algorithm 3: Private Set Union (PSU)

```
1 Cloud server partitions the full index set $\mathcal{S}$;
2 foreach client $i \in C$ do
3     Represent its real index set $\mathcal{S}^{(i)}$ as a Bloom filter $\mathbf{b}^{(i)}$;
4     Randomize $\mathbf{b}^{(i)}$ to an integer vector $\mathbf{b}'^{(i)}$ by replacing each bit
        1 in $\mathbf{b}^{(i)}$ with a random integer from $\mathbb{Z}_R$;
5     Use a bit vector $\mathbf{a}^{(i)}$ to indicate whether there exists an element
        in $\mathcal{S}^{(i)}$ falling into the partitions of $\mathcal{S}$;
6     Randomize $\mathbf{a}^{(i)}$ to an integer vector $\mathbf{a}'^{(i)}$ by replacing each bit
        1 in $\mathbf{a}^{(i)}$ with a random integer from $\mathbb{Z}_R$;
7     Submit materials for securely aggregating $\mathbf{b}'^{(i)}, \mathbf{a}'^{(i)}$;
8 Cloud server obtains $\sum_{i \in C} \mathbf{b}'^{(i)}$ and $\sum_{i \in C} \mathbf{a}'^{(i)}$ with secure
    aggregation, reconstructs the union $\bigcup_{i \in C} \mathcal{S}^{(i)}$, and delivers the
    union to each live client $i \in \hat{C}$.
```

PSU promises a wide range of applications in practice, including union queries over several databases, and, more generally, integration or sharing of datasets from multiple private sources. According to the representation format of a set, existing PSU protocols can be generally divided into two categories: polynomial-based [23, 25, 28, 29, 46] and Bloom filter-based [15, 33, 38]. For the polynomial-based protocols, elements of a set are represented as the roots of a polynomial, and the union of two sets is converted to the multiplication of two polynomials. For the protocols based on Bloom filter, the union operation over sets is normally transformed to the element-wise OR operation over the Bloom filters, and the logic OR operation can be further converted to bit addition and bit multiplication. To obliviously perform addition and multiplication operations, the two kinds of protocols mainly turn to generic secure two-party/multi-party computation, or outsource secure computation to multiple noncolluding servers. Due to unaffordable overhead, none of the existing PSU protocols have been deployed in practice. Besides inefficiency, the basic setting of these protocols significantly differs from that of FSL, where clients cannot directly communicate or natively authenticate with each other and should mediate through an untrusted cloud server. In addition, the set elements here can come from a billion-scale domain, which has not been touched in previous work as of yet.

Given the infeasibility of existing work and the atypical setting of FSL, we propose a new PSU scheme. As shown in Algorithm 3, each chosen client in one round of FSL first represents its real index set as a Bloom filter (Line 3). The details about how to set the parameters of the Bloom filter can be found in [7, 10, 47]. Then, each chosen client randomizes its Bloom filter by replacing each bit 1 with a random integer from $\mathbb{Z}_R = \{0, 1, \ldots, R-1\}$, while keeping each bit 0 unchanged (Line 4). All the chosen clients use their randomized Bloom filters rather than original ones to participate in secure aggregation (Line 7). With the secure aggregation guarantee, the untrusted cloud server obtains only the sum of the randomized Bloom filters (Line 8), but cannot know any individual randomized Bloom filter, which protects the original Bloom filter and the underlying real index set of each chosen client during the aggregation process. Further, the randomization process guarantees that the aggregate Bloom filter contains only the necessary membership information of the elements in the union and obscures unnecessary count numbers. In other words, the untrusted cloud server knows only whether there exists a client having a certain index or not, but cannot know how many clients have the index, which exactly reaches the goal of PSU. Suppose the original Bloom filters are securely aggregated without randomization. The aggregate Bloom filter is actually a counting Bloom filter, equivalent to constructing it from scratch by sequentially inserting each chosen client's real index set. Besides the desired membership information, the counting Bloom filter also contains the count numbers, which are undesired leakage in PSU. Overall, our PSU is structured internally in a new way of first local randomization and then oblivious addition, which uses the counting Bloom filter as a springboard, but conceals the undesired count numbers. Thus, our PSU avoids the costly oblivious multiplications in the common practice to derive the union of sets by performing bit-wise OR operations over the Bloom filters.

After obtaining the sum of the randomized Bloom filters, the cloud server can recover the union by doing membership tests for the full index set. It is prohibitively time-consuming if the full domain of index is huge (e.g., in the magnitude of billions in Taobao's e-commerce scenario). Even worse, the direct enumeration method can also introduce a large number of false positives to the union (i.e., those indices that are not in the union but falsely judged to be in), further leading to unnecessary redundancy in the download and upload phases. To handle these problems, we incorporate a private partition union to narrow down the scope of index for the union reconstruction above, thereby circumventing the full index set. We let the cloud server divide the full domain of index into a certain number of partitions ahead of time (Line 1). A good partition scheme needs to well balance the pros in the union reconstruction phase and the cons of additional cost. One trivial partition strategy is interval-based, where the full index set is sequentially and evenly split into some intervals. Another partition strategy is hierarchical and is often application-oriented. According to the size of partitions, we can determine the retrieval level or granularity. For example, (1) in the e-commerce recommendation, a tree structure can be used to partition the goods IDs, where the tree is initialized with the category information and further can be optimized using learning methods [57, 58]; and (2) in the natural language processing, a word semantic hierarchy can be built with the expert knowledge from WordNet [37, 39]. Given the partitions, each client first uses a bit vector to indicate whether there exists an index in its real index set falling into the partitions (Line 5). Just like hiding the concrete count numbers for the Bloom filter, the client replaces each bit 1 in its partition indicator vector with a random integer (Line 6). Then, the cloud server obtains the sum of the randomized partition indicator vectors through secure aggregation and reveals those partitions with nonzero integers in the corresponding positions (Lines 7 and 8). By simply doing membership tests for the indices falling into these partitions, the cloud server efficiently reconstructs the union and delivers it to all live clients (Line 8).

5 THEORETICAL ANALYSIS

In this section, we first analyze the security and privacy of SFSL. We then analyze its complexity and compare with SFL at the same security and privacy levels. Due to space limitations, detailed proofs and analyses are reserved in the complete manuscript [42].

5.1 Security and Privacy Analyses

Under Definition 3.1, we analyze the LDP of our index set randomization in Algorithm 2. We first prove that the union of the chosen clients' real index sets is necessary and sufficient to be the input. Then, as two stepping stones, we analyze the LDP guarantees of the permanent randomized response and a one-time instantaneous randomized response, which impose an upper bound and a lower bound on the LDP of the index set randomization, respectively.

Lemma 5.1. *The union of the chosen clients' real index sets in a certain communication round is necessary and sufficient for the index set randomization of each chosen client in the round.*

Lemma 5.2. *Permanent randomized response achieves LDP of* $\epsilon_{\infty}^{(i)} = \ln\left(\max\left(\frac{p_1^{(i)}}{p_2^{(i)}}, \frac{p_2^{(i)}}{p_1^{(i)}}, \frac{1-p_1^{(i)}}{1-p_2^{(i)}}, \frac{1-p_2^{(i)}}{1-p_1^{(i)}}\right)\right)$.

Lemma 5.3. *A one-time instantaneous randomized response achieves LDP of* $\epsilon_1^{(i)} = \ln\left(\max\left(\frac{p_5^{(i)}}{p_6^{(i)}}, \frac{p_6^{(i)}}{p_5^{(i)}}, \frac{1-p_5^{(i)}}{1-p_6^{(i)}}, \frac{1-p_6^{(i)}}{1-p_5^{(i)}}\right)\right)$.

Theorem 5.4. *When client i participates in any number of rounds, index set randomization satisfies* $\epsilon^{(i)}$*-LDP, where* $\epsilon_1^{(i)} \leq \epsilon^{(i)} \leq \epsilon_{\infty}^{(i)}$.

If client i sets $p_1^{(i)} = p_2^{(i)} = p_3^{(i)} = p_4^{(i)} = 1$, then $p_5^{(i)} = p_6^{(i)} = 1, \epsilon_1^{(i)} = 0, \epsilon_{\infty}^{(i)} = 0$, and $\epsilon^{(i)} = 0$. This corresponds to the case that any index in the union will always receive a permanent "Yes" answer and an instantaneous "Yes" answer. In other words, if client i takes the union as its randomized index set, it can achieve the strongest 0-LDP as in conventional FL.

We next analyze the stage of securely aggregating submodel updates in SFSL under Definition 3.2. We consider any index j from the union. We let $n_{j,0}$ and $n_{j,1}$ denote the numbers of live chosen clients who do not have and have j in reality, respectively. For feasibility and clarity in analysis, we let each chosen client use the same probability parameters, namely, $\forall i \in C, p_1^{(i)} = p_1, p_2^{(i)} = p_2, p_3^{(i)} = p_3, p_4^{(i)} = p_4$, which result in the same $p_5^{(i)} = p_1(p_3 - p_4) + p_4 = p_5$ and $p_6^{(i)} = p_2(p_3 - p_4) + p_4 = p_6$. Then, we have Theorem 5.5 as follows:

Theorem 5.5. *SFSL is a client-tunable privacy protection mechanism for aggregating submodel updates and guarantees that: Event 1 happens with probability* $p_7 = n_{j,1}p_5(1 - p_5)^{n_{j,1}-1}(1 - p_6)^{n_{j,0}}$*, and Event 2 happens with probability* $p_8 = (1 - p_5)^{n_{j,1}}(1 - (1 - p_6)^{n_{j,0}})$.

We still examine the case that each chosen client uses the union as its randomized index set to securely upload the submodel update by setting $p_1 = p_2 = p_3 = p_4 = 1$, resulting in $p_5 = p_6 = 1$ and $p_7 = p_8 = 0$. This is the strongest privacy for aggregating submodel updates as in SFL. Combining with the aforementioned 0-LDP for each chosen client, we can find that SFSL with the setting $p_1 = p_2 = p_3 = p_4 = 1$ (essentially, $p_5 = p_6 = 1$) is as secure as SFL under Definition 3.1 and Definition 3.2. We further generalize this observation in Theorem 5.6, which is free of any privacy definition.

Theorem 5.6. *If each chosen client uses the union of real index sets as its randomized index set, then SFSL is as secure as SFL.*

We finally analyze the security of our PSU scheme in Algorithm 3.

Theorem 5.7. *Only the union of the chosen clients' real index sets is revealed in the proposed PSU protocol.*

Table 1: Complexities of SFSL (and PSU in it) at the same security and privacy levels as SFL.

		Communication	Time	Space
Client	SFSL	$O(nsd)$	$O(n^2sd)$	$O(nsd)$
	PSU	$O(ns)$	$O(n^2s)$	$O(ns)$
	SFL	$O(n + md)$	$O(n^2 + nmd)$	$O(n + md)$
Server	SFSL	$O(n^2sd)$	$O(n^3sd)$	$O(n^2 + nsd)$
	PSU	$O(n^2s)$	$O(n^3s)$	$O(n^2 + ns)$
	SFL	$O(n^2 + nmd)$	$O(n^2md)$	$O(n^2 + md)$

*$|\bigcup_{i \in C} \mathcal{S}^{(i)}| \ll |\mathcal{S}| \Rightarrow ns \ll m$.

5.2 Complexity Analysis and Comparison

We analyze the communication, time, and space complexities of the client and the cloud server in one round of SFSL. We introduce SFL as a benchmark for comparison. To be consistent with the analysis of the secure aggregation protocol [8], we adopt the honest-but-curious setting and consider the worst-case dropouts of clients.

We first analyze SFSL. For feasibility and clarity, we let each client use the same p_1, p_2, p_3, p_4 and the resulting p_5, p_6. We assume that the expected cardinality of each client's real index set is s, which indicates that the expected cardinality of the union of n chosen clients' real index sets $\bigcup_{i \in C} \mathcal{S}^{(i)}$ is upper bounded by ns. We note that ns is normally much less than the cardinality of the full index set m (i.e., $ns \ll m$). In addition, the expected cardinality of each client's randomized index set $\mathcal{S}''^{(i)}$ is upper bounded by $sp_5 + (n-1)sp_6$, and the expected cardinality of each client's succinct index set $\mathcal{S}^{(i)} \bigcap \mathcal{S}''^{(i)}$ is sp_5. Moreover, according to [7, 10, 47], the optimal length of the Bloom filter in the proposed PSU protocol is proportional to the cardinality of the set to be filtered, namely, the cardinality of the union. In what follows, we directly show the complexity formulas containing p_5, p_6. First, the overall communication complexities of the client and the cloud server are $O(ns + (sp_5 + (n-1)sp_6)(2d+1))$ and $O(n^2s + n(sp_5 + (n-1)sp_6)(2d+1))$, respectively. Second, the overall time complexities of the client and the cloud server are $O(n^2s + n(sp_5 + (n-1)sp_6)(d+1))$ and $O(n^3s + n^2(sp_5 + (n-1)sp_6)(d+1))$, respectively. Third, the overall space complexities of the client and the cloud server are $O(ns + (sp_5 + (n-1)sp_6)(d+1))$ and $O(n^2 + ns + (sp_5 + (n-1)sp_6)(n + d + 1))$, respectively. In fact, considering secure aggregation is the most costly primitive underlying SFSL, the complexity formulas above can be roughly estimated by leveraging the complexity analysis results of the secure aggregation protocol reported in [8]. More specifically, we let the dimension of vector there take $O(ns + (sp_5 + (n-1)sp_6)(d+1))$ here, where $O(ns)$ corresponds to the optimal length of the Bloom filter and the preset constant size of the partition indicator vector in our PSU, and $(sp_5 + (n-1)sp_6)(d+1)$ accounts for the total size of the weighted submodel update and corresponding count vector with the randomized index set. From the analysis above, we can derive that our SFSL is quite scalable because its complexity depends on the cardinality of the union of n chosen clients' real index sets ns, but is independent of the cardinality of the full index set m that controls the size of the full model. In other words, SFSL completely relieves the dependence on the full model.

Table 2: Statistics of Taobao dataset.

Type	#User(s)	#Goods	#Categories	#Samples
Test (Full)	24,790	138,829	4,758	1,010,284
Train (Full)	49,023	143,534	4,815	15,854,357
Train (Per Client)	1	301	117	323

We next compare our SFSL with SFL at the same security and privacy levels by imposing $p_5 = p_6 = 1$ in SFSL. Table 1 shows their complexities along with the complexity of our PSU. The complexities of SFL and PSU are derived using the complexity analysis of the secure aggregation protocol, where the dimension of vector takes the size of the full model md and $O(ns)$, respectively. From Table 1, we can draw that given $ns \ll m$ in our application scenario, the complexities of the client and the cloud server in SFSL are both far lower than those in SFL. We can also draw that only if $ns < m$, our SFSL outperforms SFL. In the worst case where $ns = m$ (i.e., the union of the chosen clients' real index sets is the full index set), the complexity of SFSL is the same as that of SFL. An intuitive explanation is that if all the clients participate in each round of FSL, the union of the chosen clients' real index sets is the full index set, and SFSL degenerates to SFL. However, the full participation is ideal and impractical because of the unavailability of some clients in each round. Thus, the worst case may never happen. We note that ns cannot be larger than m, because the full index set is actually the union of all the clients' real index sets. In summary, SFSL outperforms SFL in any scale of FSL applications, where ubiquitous data heterogeneity results in model differentiation for the clients.

6 EVALUATION

Dataset: We use an industrial dataset[1] built from 30-day impression and click logs of Taobao users from June 15, 2019 to July 15, 2019. For a certain Taobao user, we leverage his/her click behaviors in previous 14 days as historical data to predict his/her click and non-click behaviors in the following 1 day. We leave out the behaviors within the last 1 day as the target items of the test set while putting the other samples into the training set. Specifically, the test set is located on the cloud server to judge the accuracy and convergency of the global model. For the full training set, we further cluster each Taobao user's data as a training set located on a client. Table 2 shows some statistics about the dataset.

Model, Hyperparameters, and Metric: We take the deployed DIN in Alibaba [56] as the recommendation model for FSL, where the number of columns in the embedding matrix is set to 18. Except the embedding layer for user IDs, goods IDs, and category IDs, the parameters of the other network layers in DIN, including the attention layer and the fully connected layer, are of size 64,327. Hence, the global model on the cloud server is of size 3,617,023, whereas a desired submodel on the client is of size 71,869 in average, which is only 1.99% of the global model's size and roughly requires 0.27MB space using 32-bit representation. For each client's local training, we choose mini-batch *stochastic gradient descent (SGD)* as the optimization algorithm, set the batch size to 2, and set the local epoch number to 1. In addition, we initially set the learning rate to 1 and further apply exponential decay with the decay rate of 0.999 per communication round. For the cloud server's global testing, we adopt a golden metric in the task of *click-through rate (CTR)* prediction, called AUC, and set the batch size to 1,024.

Table 3: Choices of probability parameters (CPPs) and resulting privacy levels. CPP1 corresponds to pure FSL with secure aggregation. CPP5 is as secure as SFL.

	p_1, p_3	p_2, p_4	p_5	p_6	ϵ_1	ϵ_∞	p_7	p_8
CPP1	1	0	1	0	∞	∞	86.7%	0
CPP2	15/16	1/16	88.3%	11.7%	2.02	2.71	0	10.3%
CPP3	7/8	1/8	78.1%	21.9%	1.27	1.95	0	19.5%
CPP4	3/4	1/4	62.5%	37.5%	0.51	1.10	0	34.2%
CPP5	1	1	1	1	0	0	0	0

* Smaller ϵ_1, ϵ_∞, p_7, p_8 indicate better privacy for each chosen client i.

Prototypes and Configurations: We implemented the prototypes of our SFSL and the baseline SFL in Python 2.7.16, and the source code is publicly available from [41]. Figure 2 provides an overview of SFSL from the perspective of a client in a certain round. We took a synchronous architecture on top and implemented a communication module between the cloud server and each client with standard socket programming. We used TensorFlow 1.12.0 to implement DIN. We mainly used PyCryptodome 3.7.3 to implement the secure aggregation protocol in [8]. To support modular addition underlying secure aggregation, we performed float-to-integer conversion for each client's submodel or full model update with stochastic quantization [48], where the quantization level is set to 2^{15}. We let each client use the same *choice of probability parameters (CPP)* in SFSL. Table 3 lists 5 CPPs in the evaluation and their resulting privacy levels of index set randomization and aggregating submodel updates when the number of chosen clients per round n is 100. Specifically, CPP1 corresponds to directly combining FSL with secure aggregation, which lets each client reveal its real index set and provides the worst LDP. The resulting $p_7 = 86.7\%$ at CPP1 further implies that 86.7% of the goods IDs in the union of 100 randomly chosen clients' real goods IDs involve a single client. Therefore, the user data in Taobao's e-commerce context are highly heterogeneous. In contrast, CPP5 lets each client use the union of the chosen clients' real index sets as its randomized index set. By Theorem 5.6, SFSL with CPP5 is as secure as SFL.

Our running environment is a Linux workstation with 64-bit Ubuntu 18.04.2 OS. The processor is Intel(R) Core(TM) i9-9900K with 8 cores, the base frequency is 3.60GHz, the memory size is 64GB, and the cache size is 16MB. The workstation is also equipped with 2 NVIDIA's GeForce RTX 2080 Ti graphics cards. To manifest the difference between the roles of client and cloud server, (1) from hardware, we ran all the clients only on CPU, but allowed the cloud server to accelerate its operations using GPU; and (2) from parallelism and concurrency, we optimized the cloud server's hotspot functions with Python's multiprocessing library. For more implementation details, please refer to the complete manuscript [42].

6.1 Model Accuracy and Convergency

We bring in centralized training and conventional FL as two baselines. We plot the AUCs of our SFSL and two baselines in Figure 3.

[1] A similar Taobao dataset is online available from Tianchi [3].

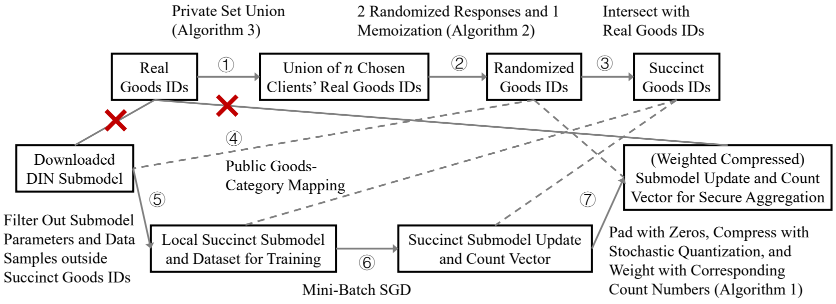


Figure 2: An implementation overview of SFSL from a client (i.e., a Taobao user) in a certain round for DIN-based CTR prediction.

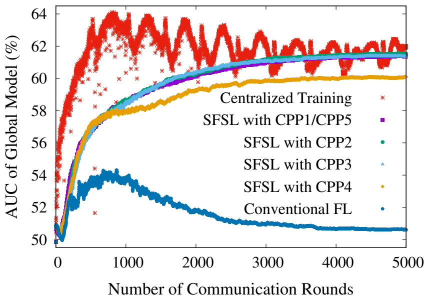
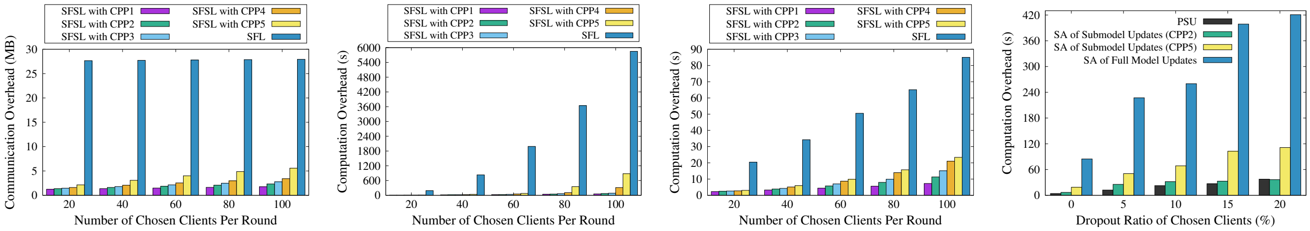


Figure 3: AUCs of the global model in centralized training, SFSL, and FL.



(a) Per Client (Overall)

(b) Per Client (Overall)

(c) Cloud Server (Overall)

(d) Cloud Server (SA)

Figure 4: Communication and computation overhead of the client and the cloud server per round in our SFSL and the baseline SFL. In Figure 4(d), SA denotes secure aggregation, and 100 clients are chosen per round.

The number of clients randomly chosen in each round n is set to 100. In addition, centralized training refers to the traditional case that the cloud server first collects data from all the clients, then trains the DIN model, and tests the global model once training over the samples with a size similar to the total size of n clients' datasets.

From Figure 3, we can see that compared with centralized training, which reaches the best AUC of 64.11% in 803 rounds, our SFSL with CPP2 achieves the highest AUC of 61.54% in 4,908 rounds, decreasing by 2.57% of AUC. In contrast, conventional FL performs worst, achieving the highest AUC of 54.32% in 867 rounds and diverging in the end. The major reason is that FL coarsely computes the weighted average of the chosen clients' full model updates proportional to their training set sizes, no matter whether one client's whole training set actually involves some model parameters (the full model excluding its desired submodel, e.g., some embedding vectors in DIN) or not, thereby inaccurately counting in the weights (i.e., the training set sizes) of those clients who contribute zero/no updates for these model parameters. With a higher heterogeneity of user data and a higher differentiation of submodels, the roughness and inaccuracy of FL will be exposed more completely, which clarifies why FL can work in the natural language context with a 10,000-word vocabulary considered by Google, but does not work well in Alibaba's e-commerce context with billion-scale goods IDs.

From Figure 3, we can also observe that CPP4 is the worst among all CPPs and achieves the highest AUC of 60.11%, because p_5, which controls the size of each client's succinct training set as well as the length of historical goods and category IDs in each training sample, in CPP4 is the smallest among all CPPs. This still explains another observation that CPP1 and CPP5 with the same $p_5 = 1$ have identical model performance.

6.2 Communication Overhead

We show the communication overhead of our SFSL and introduce SFL as a baseline. Figure 4(a) plots the overall communication overhead per client per round. We do not plot the communication overhead of the cloud server, since it is equal to the communication overhead per client multiplying by n. In more detail, the incoming data of the cloud server are exactly the total outgoing data of all n chosen clients, and vice verse. We also do not plot for different dropout ratios because this factor has little impact on the communication overhead.

One key observation from Figure 4(a) is that compared with SFL, our SFSL can sharply reduce the communication overhead. In particular, when $n = 100$, the overall communication overhead per client per round is 1.76MB, 2.33MB, 2.78MB, 3.40MB, and 5.57MB in SFSL with CPP1, CPP2, CPP3, CPP4, and CPP5, respectively, reducing 93.72%, 91.65%, 90.06%, 87.81%, and 80.05% than SFL, which incurs 27.94MB. Considering SFSL with CPP5 is as secure as SFL, we can draw that our SFSL can reduce communication overhead even with not sacrificing any security or privacy. These results coincide with the complexity analysis in Section 5.2 and Table 1.

The second key observation from Figure 4(a) is that the communication overhead per client in SFSL increases with n for a certain CPP and also increases with the serial number of CPP for a certain n. We clarify the reasons by the client's communication complexity $O(ns + (sp_5 + (n-1)sp_6)(2d+1))$ in Section 5.2. On the one hand, the communication complexity grows linearly with n. On the other hand, it is increasing with p_5 and p_6, and thus, CPP5 is the most communication expensive. Additionally, given $p_5 + p_6 = 1$ for CPPs from CPP1 to CPP4, we can simplify the communication complexity to $O(ns + (s + (n-2)sp_6)(2d+1))$, which increases with p_6 as $n > 2$.

From Table 3, we can see that p_6 increases with the serial number of CPP, implying higher communication overhead as depicted in Figure 4(a). Intuitively, p_6 controls the size of the redundant/zero parameters to be downloaded and securely uploaded and dominates the holistic trend of the communication overhead.

We finally present the communication overhead per client per round incurred by our PSU, which increases linearly with n, roughly with an increase of 0.07MB per 20 clients. In addition, our PSU is quite communication efficient and incurs 0.91MB as n reaches 100.

6.3 Computation Overhead

We now report the practical computation overhead, mainly by investigating the effects of n, the CPP, and the ratio of dropout. To be consistent with the time complexity analysis, the computation overhead includes only the run time of the client or the cloud server in executing the protocol, but ignores synchronization delay and the time overhead of testing global model. Given mobile devices are highly parallel in practice, the total run time per communication round can be estimated by adding up the computation overhead of the client and the cloud server reported here. In addition, testing the global model per round costs the cloud server 32.12s.

We first show the overall computation overhead of the client and the cloud server per round in our SFSL with different CPPs in Figure 4(b) and Figure 4(c). We still introduce SFL as a baseline. First, we can see that our SFSL significantly outperforms SFL on both sides of the client and the cloud server. In particular, when $n = 100$, at the same security and privacy levels, SFSL with CPP5 reduces 85.02% and 72.51% of the computation overhead than SFL on the client and the cloud server, respectively. When security and privacy become weaker, the advantages of our SFSL are more evident under CPP1 to CPP4. For example, CPP2 reduces 98.77% and 86.70% of the run time than SFL on the client and the cloud server, respectively. Second, we focus on a certain side, either the client or the cloud server, and can observe that its computation overhead grows with n or the serial number of CPP. The reason can be explained by the detailed time complexity in Section 5.2, while the intuition is analogous to that behind the communication overhead.

We next investigate the effect of the dropout ratio and depict the evaluation results per round in Figure 4(d) as $n = 100$. We mainly focus on the secure aggregation-based stages while ignoring the other stages irrelevant with dropout. We consider that clients are randomly selected to go offline at random points. We report only the computation overhead of the cloud sever because dropped clients do not introduce additional operation cost for live clients. From Figure 4(d), we can see that with the growth of the dropout ratio, the computation overhead of the cloud server increases. This is because the cloud server needs to remove the mutual masks between dropped and live clients in secure aggregation. We also compare our SFSL with SFL at the stage of securely aggregating submodel or full model updates. We can find that SFSL significantly outperforms SFL at this stage for any dropout ratio. Specifically, when the dropout ratio is 20%, CPP2 and CPP5 reduce 91.33% and 73.55% of the computation overhead, respectively. We finally examine our PSU and can see that it is quite efficient, even when the dropout ratio is high. In particular, the computation overhead of the cloud server is 37.66s as the dropout ratio reaches 20%.

6.4 Memory and Disk Loads

We present the practical storage overhead of our SFSL and the baseline SFL. First is about the memory overhead. The cloud server requires the video memory of 551MB, mainly for testing the global model at the end of each communication round, which is the same for all schemes. In addition, when $n = 100$, and there is no dropout, the memory overhead per client is 209MB and 281MB in SFSL with CPP2 and CPP5, respectively, reducing 59.40% and 45.43% than SFL. Correspondingly, the memory overhead of the cloud server is 1.58GB and 3.15GB, respectively, reducing 81.88% and 63.77% than SFL. Regarding the disk load, only the client in SFSL with CPP2, CPP3, and CPP4 needs to store its permanent noisy answers in the index set randomization, which roughly occupies the disk space of 280KB within the total 5,000 rounds.

6.5 Discussion on Billion-Scale Issues

We finally discuss the issues as the size of the full index set (which controls the size of the full model) and the total number of clients scale further to billions in practice. First, the baseline SFL, depending on the full model, will be prohibitively inefficient to be applicable. Second, we analyze our SFSL. SFSL relives the dependence on the full model, executes in communication rounds, and involves only a constant number of (e.g., $n = 100$) clients in each round. Thus, no additional overhead will be incurred in SFSL due to the scale of billions. We further consider how to shorten the periods of building/polishing the full model and covering the whole clients without affecting any client's efficiency and privacy. It is trivial to scale up SFSL at the level of the cloud server. In particular, a master server manages multiple child servers. For every round, each child server selects n clients, lets them use SFSL to download submodels, and securely aggregates their submodel updates with SFSL. The master server further aggregates the aggregated submodel updates from the child servers to update the global model.

7 CONCLUSION

In this paper, we have proposed a submodel design SFSL for mobile clients to effectively and efficiently train large-scale models under the coordination of an untrusted cloud server while keeping user data private. We further have applied SFSL to the e-commerce recommendation scenario of Alibaba, implemented a prototype system, and validated its practical feasibility over a Taobao dataset.

ACKNOWLEDGMENTS

We sincerely thank our anonymous shepherd and reviewers for their insightful feedback. We also thank Renjie Gu, Hongtao Lv, Hejun Xiao, and Zhenzhe Zheng, as well as many colleagues in Alibaba, for meaningful discussions and great engineering support. This work was supported in part by National Key R&D Program of China No. 2019YFB2102200, China NSF grant No. 61972252, 61972254, 61672348, and 61672353, Joint Scientific Research Foundation of the State Education Ministry No. 6141A02033702, and Alibaba Innovation Research (AIR) Program. The opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies or the government. Fan Wu (fwu@cs.sjtu.edu.cn) is the corresponding author of this paper.

REFERENCES

[1] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proc. of CCS.* ACM, 308–318.

[2] Naman Agarwal, Ananda Theertha Suresh, Felix Yu, Sanjiv Kumar, and H. Brendan McMahan. 2018. cpSGD: Communication-efficient and differentially-private distributed SGD. In *Proc. of NeurIPS.* 7575–7586.

[3] Alimama. 2017. Ad Display/Click Data on Taobao.com. https://tianchi.aliyun.com/dataset/dataDetail?dataId=56.

[4] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. In *Proc. of S&P.* IEEE, 962–979.

[5] Apple's Differential Privacy Team. 2017. Learning with Privacy at Scale. *Apple Machine Learning Journal* 1, 8 (2017).

[6] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How To Backdoor Federated Learning. In *Proc. of AISTATS.* PMLR, 2938–2948.

[7] Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM* 13, 7 (1970), 422–426.

[8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proc. of CCS.* ACM, 1175–1191.

[9] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. 2005. Evaluating 2-DNF Formulas on Ciphertexts. In *Proc. of TCC.* Springer, 325–341.

[10] Andrei Broder and Michael Mitzenmacher. 2004. Network Applications of Bloom Filters: A Survey. *Internet Mathematics* 1, 4 (2004), 485–509.

[11] Qingqing Cao, Noah Weber, Niranjan Balasubramanian, and Aruna Balasubramanian. 2019. DeQA: On-Device Question Answering. In *Proc. of MobiSys.* ACM, 27–40.

[12] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In *Proc. of CCS.* ACM, 1223–1237.

[13] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. 2019. Federated Learning Of Out-Of-Vocabulary Words. arXiv: 1903.10635. http://arxiv.org/abs/1903.10635.

[14] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. 2019. Behavior Sequence Transformer for E-commerce Recommendation in Alibaba. arXiv: 1905.06874. http://arxiv.org/abs/1905.06874.

[15] Alex Davidson and Carlos Cid. 2017. An Efficient Toolkit for Computing Private Set Operations. In *Proc. of ACISP.* Springer, 261–278.

[16] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting Telemetry Data Privately. In *Proc. of NeurIPS.* 3574–3583.

[17] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407.

[18] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. 2019. Amplification by Shuffling: From Local to Central Differential Privacy via Anonymity. In *Proc. of SODA.* ACM-SIAM, 2468–2479.

[19] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proc. of CCS.* ACM, 1054–1067.

[20] European Parliament and Council of the European Union. 2016. The General Data Protection Regulation (EU) 2016/679 (GDPR). https://eur-lex.europa.eu/eli/reg/2016/679/oj. Took effect from May 25, 2018.

[21] Giulia Fanti, Vasyl Pihur, and Úlfar Erlingsson. 2016. Building a RAPPOR with the Unknown: Privacy-Preserving Learning of Associations and Data Dictionaries. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2016, 3 (2016), 41–61.

[22] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. 2014. Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing. In *Proc. of USENIX Security.* 17–32.

[23] Keith Frikken. 2007. Privacy-Preserving Set Union. In *Proc. of ACNS.* Springer, 237–252.

[24] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated Learning for Mobile Keyboard Prediction. arXiv: 1811.03604. http://arxiv.org/abs/1811.03604.

[25] Jeongdae Hong, Jung Woo Kim, Jihye Kim, Kunsoo Park, and Jung Hee Cheon. 2013. Constant-round privacy preserving multiset union. *Bulletin of the Korean Mathematical Society* 50, 6 (2013), 1799–1816.

[26] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2019. Advances and Open Problems in Federated Learning. arXiv: 1912.04977. http://arxiv.org/abs/1912.04977.

[27] Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2008. What Can We Learn Privately?. In *Proc. of FOCS.* IEEE, 531–540.

[28] Lea Kissner and Dawn Xiaodong Song. 2005. Privacy-Preserving Set Operations. In *Proc. of CRYPTO.* Springer, 241–257.

[29] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. 2019. Scalable Private Set Union from Symmetric-Key Techniques. IACR Cryptology ePrint Archive, Report 2019/776. https://eprint.iacr.org/2019/776.

[30] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. arXiv: 1610.05492. http://arxiv.org/abs/1610.05492.

[31] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.

[32] Fuyu Lv, Taiwei Jin, Changlong Yu, Fei Sun, Quan Lin, Keping Yang, and Wilfred Ng. 2019. SDM: Sequential Deep Matching Model for Online Large-scale Recommender System. In *Proc. of CIKM.* ACM, 2635–2643.

[33] Dilip Many, Martin Burkhart, and Xenofontas Dimitropoulos. 2012. *Fast private set operations with sepia.* Technical Report TIK-Report No. 345. Communication Systems Group, ETH Zürich, Switzerland.

[34] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proc. of AISTATS.* PMLR, 1273–1282.

[35] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning Differentially Private Recurrent Language Models. In *Proc. of ICLR.* OpenReview.net.

[36] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. 2019. Exploiting Unintended Feature Leakage in Collaborative Learning. In *Proc. of S&P.* IEEE, 497–512.

[37] George A. Miller. 1995. WordNet: A Lexical Database for English. *Communications of the ACM* 38, 11 (1995), 39–41.

[38] Atsuko Miyaji and Katsunari Shishido. 2018. Efficient and Quasi-accurate Multiparty Private Set Union. In *Proc. of SMARTCOMP.* IEEE, 309–314.

[39] Frederic Morin and Yoshua Bengio. 2005. Hierarchical Probabilistic Neural Network Language Model. In *Proc. of AISTATS.* Society for Artificial Intelligence and Statistics, 246–252.

[40] M. Nasr, R. Shokri, and A. Houmansadr. 2019. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In *Proc. of S&P.* IEEE, 1021–1035.

[41] Chaoyue Niu, Renjie Gu, Hongtao Lv, and Hejun Xiao. 2020. Source code for secure federated submodel learning. https://github.com/NiuChaoyue/Secure-Federated-Submodel-Learning.

[42] Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. 2019. Secure Federated Submodel Learning. arXiv: 1911.02254. http://arxiv.org/abs/1911.02254.

[43] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proc. of EUROCRYPT.* Springer, 223–238.

[44] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. 2018. Private Stateful Information Retrieval. In *Proc. of CCS.* ACM, 1002–1019.

[45] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. 2019. Federated Learning for Emoji Prediction in a Mobile Keyboard. arXiv: 1906.04329. http://arxiv.org/abs/1906.04329.

[46] Jae Hong Seo, Jung Hee Cheon, and Jonathan Katz. 2012. Constant-Round Multiparty Private Set Union Using Reversed Laurent Series. In *Proc. of PKC.* Springer, 398–412.

[47] David Starobinski, Ari Trachtenberg, and Sachin Agarwal. 2003. Efficient PDA Synchronization. *IEEE Transactions on Mobile Computing* 2, 1 (2003), 40–51.

[48] Ananda Theertha Suresh, Felix X. Yu, Sanjiv Kumar, and H. Brendan McMahan. 2017. Distributed Mean Estimation with Limited Communication. In *Proc. of ICML.* PMLR, 3329–3337.

[49] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba. In *Proc. of KDD.* ACM, 839–848.

[50] Tianhao Wang, Bolin Ding, Jingren Zhou, Cheng Hong, Zhicong Huang, Ninghui Li, and Somesh Jha. 2019. Answering Multi-Dimensional Analytical Queries under Local Differential Privacy. In *Proc. of SIGMOD.* ACM, 159–176.

[51] Tianhao Wang, Ninghui Li, and Somesh Jha. 2018. Locally Differentially Private Frequent Itemset Mining. In *Proc. of S&P.* IEEE, 127–143.

[52] Stanley L. Warner. 1965. Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias. *J. Amer. Statist. Assoc.* 60, 309 (1965), 63–69.

[53] Xiufeng Xie and Kyu-Han Kim. 2019. Source Compression with Bounded DNN Perception Loss for IoT Edge Computer Vision. In *Proc. of MobiCom*. ACM, 47:1–47:16.

[54] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Transactions on Intelligent Systems and Technology* 10, 2 (2019), 12:1–12:19.

[55] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep Interest Evolution Network for Click-Through Rate Prediction. In *Proc. of AAAI*. AAAI Press, 5941–5948.

[56] Guorui Zhou, Xiaoqiang Zhu, Chengru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep Interest Network for Click-Through Rate Prediction. In *Proc. of KDD*. ACM, 1059–1068.

[57] Han Zhu, Daqing Chang, Ziru Xu, Pengye Zhang, Xiang Li, Jie He, Han Li, Jian Xu, and Kun Gai. 2019. Joint Optimization of Tree-based Index and Deep Model for Recommender Systems. In *Proc. of NeurIPS*. 3973–3982.

[58] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning Tree-based Deep Model for Recommender Systems. In *Proc. of KDD*. ACM, 1079–1088.

[59] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep Leakage from Gradients. In *Proc. of NeurIPS*. 14774–14784.