Private Federated Submodel Learning with Sparsification

Sajani Vithana Sennur Ulukus
Department of Electrical and Computer Engineering
University of Maryland, College Park, MD 20742
spallego@umd.edu *ulukus@umd.edu*

***Abstract*—We investigate the problem of private read update write (PRUW) in federated submodel learning (FSL) with sparsification. In FSL, a machine learning model is divided into multiple submodels, where each user updates only the submodel that is relevant to the user's local data. PRUW is the process of privately performing FSL by reading from and writing to the required submodel without revealing the submodel index or the values of updates to the databases. Sparsification is a widely used concept in learning, where the users update only a small fraction of parameters to reduce the communication cost. Revealing the coordinates of these selected (sparse) updates leaks privacy of the user. We show how PRUW in FSL can be performed with sparsification. We propose a novel scheme which privately reads from and writes to arbitrary parameters of any given submodel, without revealing the submodel index, values of the updates, or the coordinates of the sparse updates, to databases. The proposed scheme achieves significantly lower reading and writing costs compared to what is achieved without sparsification.**

I. Introduction

Federated learning (FL) [1]–[4] enables distributed machine learning without the users having to share their private data directly with a central server. This solves user privacy concerns to a certain extent and decentralizes the processing power requirements. However, the communication cost of FL is considerable as a large number of users keep sharing gradient updates with the central server in an iterative manner. Gradient sparsification [5]–[13], gradient quantization [14]–[16], and federated submodel learning (FSL) [17]–[23] are some of the solutions proposed for this problem. In this work, we propose a novel scheme that combines FSL and gradient sparsification while preserving information-theoretic privacy of the users.

In FSL, a machine learning model is divided into multiple submodels based on different types of data used to train parts of the model. In FSL, a given user downloads (reads) and updates (writes) only the submodel that is relevant to the user's local data, which reduces the communication cost. In this process, the updated submodel index as well as the values of the updates leak information about the type of data that the user has. Thus, in order to perform FSL while preserving user privacy, both the submodel index and the values of the updates need to be kept private in both reading and writing phases. This is known as private read update write (PRUW) [17]–[23]. The reading phase of PRUW is similar to private information retrieval (PIR) [24]–[38].

This work was supported by ARO Grant W911NF2010142, and NSF Grants CCF 17-13977 and ECCS 18-07348.

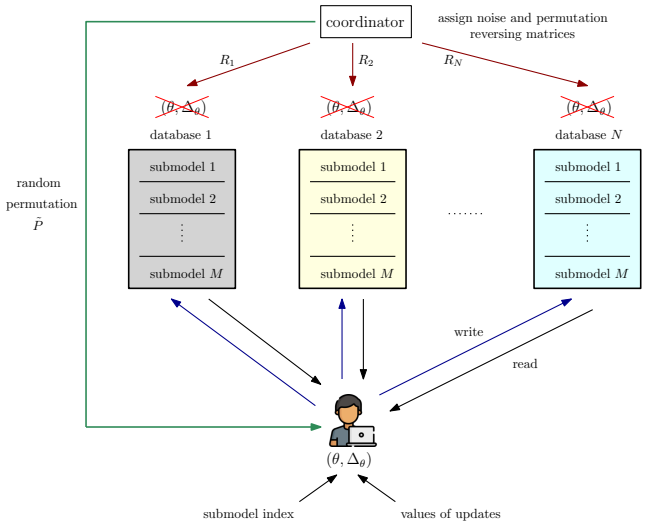

Fig. 1. System model.

Sparsification is a widely used technique in FL, where users send (receive) only a small fraction of updates (parameters) to the servers (from databases) in order to reduce the communication cost. In this process, the users send only a selected set of updates (randomly chosen or largest amplitude values) along with their positions to the servers. However, directly sending the updates and their positions leaks user's privacy [39]–[56]. In this paper, we propose a method to privately send the sparse set of updates along with their indices using a noisy shuffling mechanism, in the setting of private FSL.

PRUW in FSL with sparsification consists of two phases, the reading phase in which a user privately downloads the required submodel, and the writing phase in which the user privately uploads the updates back to the databases. In the writing phase, each user only uploads a small fraction of updates to reduce the writing cost. In the reading phase, each user only downloads a selected set of parameters of the required submodel, determined by the databases, e.g., the sparse set of parameters that were updated by all users in the previous iteration. In this process, the updating submodel index and the values of the updates are kept private. Thus, we need to ensure that the actual positions of the sparse updates are not directly sent to the databases since they reveal the values (zero) of those updates whose indices were not specified. To achieve this, we propose a method which is an extension of the PRUW scheme in [18]. The system model, as in [18] and

[19], includes a coordinator that helps initialize the PRUW process. The coordinator in [18], [19] is necessary to initialize the same noise terms in the storage across the N databases. In this work, the coordinator is used again to introduce a shuffling mechanism that hides the real positions of the sparse updates from databases to guarantee privacy in FSL with sparsification.

The major contributions of this work are: 1) introduction of the concept of sparsification in FSL to achieve significantly lower communication costs while still achieving information-theoretic privacy, 2) a novel scheme that performs PRUW with arbitrary sparsification rates with zero information leakage.

II. Problem Setting

Consider M independent submodels, each having P subpackets, stored in N non-colluding databases. At a given time instance t, a user reads, updates and writes to one of the M submodels, while not revealing any information about the updated submodel index or the values of updates. The storage, queries and updates consist of symbols from a finite field $\mathbf{F}_q$. In this work, we consider top r sparsification, where the users only upload the most significant r, $0 \leq r \leq 1$ fraction of updates of the updating submodel in the writing phase, and only download an r', $0 \leq r' \leq 1$ fraction of parameters of the required submodel, determined by the databases in the reading phase, to reduce the communication cost.[1] Precisely, the users only update Pr most significant subpackets in the writing phase and download only Pr' subpackets in the reading phase.

The reduction in the communication cost of PRUW with sparsification results from communicating the non-zero-valued updates (selected parameters) and their positions to the databases (users) in the writing (reading) phase. However, this leaks information about $1-r$ of the updates in the writing phase, as their values (zero) are revealed to the databases.[2] To avoid this, we use a shuffling method, in which actual positions of the non-zero updates are randomly shuffled with the aid of a coordinator as shown in Fig. 1. This ensures that the indices of zero-valued parameters are hidden from the databases.

The three components in private FSL with sparsification that need to be kept private are: 1) index of the submodel updated by each user, 2) values of the updates, and 3) indices of the sparse updates. Note that 3 is a requirement that is implied by 2. The formal descriptions of the privacy constraints are given below. The constraints are presented in the perspective of a single user at time t, even though multiple users update the model simultaneously.

Privacy of the submodel index: No information on the index of the submodel that is being updated, θ, is allowed to leak to any of the databases, i.e., for each n,

$$I(\theta^{[t]}; Q_n^{[t]}, Y_n^{[t]} | Q_n^{[1:t-1]}, Y_n^{[1:t-1]}, S_n^{[0:t-1]}) = 0, \quad (1)$$

where Q_n and Y_n are the queries and updates/coordinates sent by a given user to database n in the reading and writing phases, at corresponding time instances indicated in square brackets and S_n is the storage of database n.

Privacy of the values of updates: No information on the values of the updates is allowed to leak to any of the databases, i.e., for each n, $n \in \{1, \ldots, N\}$,

$$I(\Delta_\theta^{[t]}; Q_n^{[t]}, Y_n^{[t]} | Q_n^{[1:t-1]}, S_n^{[0:t-1]}) = 0 \quad (2)$$

where $\Delta_\theta^{[t]}$ is the update of submodel θ generated at time t.

Security of submodels: No information on the submodels is allowed to leak to any of the databases, i.e., for each n,

$$I(W_{1:M}^{[t]}; S_n^{[t]}) = 0, \quad (3)$$

where $W_k^{[t]}$ is the kth submodel at time t.

Correctness in the reading phase: The user should be able to correctly decode the sparse set of subpackets (denoted by J) of the required submodel in the reading phase, i.e.,

$$H(W_{\theta,J}^{[t-1]} | Q_{1:N}^{[t]}, A_{1:N}^{[t]}, \theta) = 0, \quad (4)$$

where $W_{\theta,J}^{[t-1]}$ is the set of subpackets in set J of submodel W_θ at time $t-1$ and $A_n^{[t]}$ is the answer from database n.

Correctness in the writing phase: Let θ be the updating submodel index and J' be the set of most significant Pr subpackets of W_θ updated by a given user. Then, the subpacket s of submodel m at time t given by $W_m^{[t]}(s)$ is updated as,

$$W_m^{[t]}(s) = \begin{cases} W_m^{[t-1]}(s) + \Delta_m^{[t]}(s), & \text{if } m = \theta \text{ and } s \in J' \\ W_m^{[t-1]}(s), & \text{if } m \neq \theta \text{ or } s \notin J' \end{cases}, \quad (5)$$

where $\Delta_m^{[t]}(s)$ is the corresponding update of $W_m^{[t-1]}(s)$.

Reading and writing costs: The reading and writing costs are defined as $C_R = \frac{\mathcal{D}}{L}$ and $C_W = \frac{\mathcal{U}}{L}$, respectively, where $\mathcal{D}$ is the total number of bits downloaded in the reading phase, $\mathcal{U}$ is the total number of bits uploaded in the writing phase, and L is the size of each submodel. The total cost C_T is the sum of the reading and writing costs, $C_T = C_R + C_W$.

III. Main Result

Theorem 1 *In a private FSL setting with N databases, M submodels, P subpackets in each submodel, and r and r' sparsification rates in the uplink and downlink, respectively, the following reading and writing costs are achievable,*

$$C_R = \frac{4r' + \frac{4}{N}(1+r')\log_q P}{1 - \frac{2}{N}}, \quad (6)$$

$$C_W = \frac{4r(1+\log_q P)}{1 - \frac{2}{N}}. \quad (7)$$

The proof of Theorem 1 is presented in Section IV.

Remark 1 *If sparsification is not considered in the PRUW process, the lowest achievable reading and writing costs are given by $C_R = C_W = \frac{2}{1-\frac{2}{N}}$; see [18]. Therefore, sparsification with smaller values of r and r' results in significantly reduced communication costs as shown in* (6) *and* (7)*.*

[1] In model training, users typically work in continuous fields and make $1-r$ of the updates equal to zero (i.e., not update) based on the concept of sparsification. These updates are then converted to symbols in $\mathbf{F}_q$. We assume that the zeros in the continuous field are converted to zeros in the finite field.

[2] Note that no information is leaked by the sparse subpacket indices in the reading phase, since the databases determine these sparse indices with no additional information from the users.

IV. Proposed Scheme

The scheme is similar to what is presented in [18] with the additional component of sparse uploads and downloads. In the writing (reading) phase of the scheme in [18], the updates (values) of all parameters in a given subpacket are combined into a single bit. Thus, a user sends (receives) P bits per database, where P is the number of subpackets in a submodel. In this work, we assume that the user only updates $Pr \ll P$, and downloads only $Pr' \ll P$ of the subpackets due to sparsification. However, revealing the indices of the subpackets with no update (all zeros) in the writing phase leaks privacy, as the values of those updates (zero) are directly known by the databases. Thus, we use a random permutation technique to hide the subpackets with zero updates.

A. Storage and Initialization

The storage of a single subpacket in database n is,

$$S_n = \begin{bmatrix} \begin{bmatrix} W_{1,1} + (f_1 - \alpha_n) \sum_{i=0}^{2\ell} \alpha_n^i Z_{1,i}^{[1]} \\ \vdots \\ W_{M,1} + (f_1 - \alpha_n) \sum_{i=0}^{2\ell} \alpha_n^i Z_{M,i}^{[1]} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} W_{1,\ell} + (f_\ell - \alpha_n) \sum_{i=0}^{2\ell} \alpha_n^i Z_{1,i}^{[\ell]} \\ \vdots \\ W_{M,\ell} + (f_\ell - \alpha_n) \sum_{i=0}^{2\ell} \alpha_n^i Z_{M,i}^{[\ell]} \end{bmatrix} \end{bmatrix}, \quad (8)$$

where the subpacketization $\ell = \frac{N-2}{4}$, $W_{i,j}$ is the jth bit of the given subpacket of submodel i, W_i, $Z_{i,j}^{[k]}$ is the $(j+1)$st noise term for the kth bit of W_i, and $\{f_i\}_{i=1}^{\ell}$, $\{\alpha_n\}_{n=1}^{N}$ are globally known distinct constants chosen from $\mathbf{F}_q$, such that each α_n and $f_i - \alpha_n$ for all $i \in \{1, \dots, \ell\}$ and $n \in \{1, \dots, N\}$ are coprime with q. The number of subpackets is $P = \frac{4L}{N-2}$.

In PRUW, at time $t = 0$, it should be ensured that all noise terms in storage are the same in all databases. This is handled by the coordinator in Fig. 1. We make use of this coordinator again in PRUW with sparsification as follows. In the reading and writing phases of PRUW with sparsification, the user only reads and writes parameters/updates corresponding to a subset of subpackets ($\ll P$) without revealing their true indices. The coordinator is used to privately shuffle the true non-zero subpacket indices as explained next.

At the beginning of the FSL system design, $t = 0$, the coordinator picks a random permutation of indices $\{1, \dots, P\}$ out of all $P!$ options, denoted by $\tilde{P}$, where P is the number of subpackets. The coordinator sends $\tilde{P}$ to all users involved in the PRUW process. Then, the coordinator sends the corresponding noise added permutation-reversing matrix to database n, $n \in \{1, \dots, N\}$, given by,

$$R_n = R + \prod_{i=1}^{\ell} (f_i - \alpha_n) \bar{Z}, \quad (9)$$

where R is the permulation-reversing matrix and $\bar{Z}$ is a random noise matrix, both of size $P \times P$. For each database, R_n is a random noise matrix, from which nothing can be learnt about the random permutation. The matrix R_n is fixed at database n at all time instances.

B. Reading Phase at Time t

The process of reading (downlink) a subset of parameters of a given submodel without revealing the submodel index or the parameter indices within the submodel is explained in this section. Let $\tilde{V}$ be the set of permuted indices of the sparse subpackets (of all submodels) to be sent to the users in the reading phase at time t. $\tilde{V}$ is determined by the databases with no additional information from the users, using an arbitrary downlink sparsification protocol. For example, in a case where the PRUW process is carried out in an iterative manner, a given user at time t must only download the subpackets that were updated in the writing phase at time $t-1$. In this case, $\tilde{V}$ is the union of all sets of permuted indices of the sparse subpackets updated by all users at time $t-1$.[3] One designated database sends $\tilde{V}$ to each user at time t. Once the users know the subset of permuted subpacket indices in $\tilde{V}$, the users can find the corresponding real subpacket indices (permutation-reversed) from the permutation $\tilde{P}$ received by the coordinator at the initialization stage. The next steps of the reading phase at time t are as follows:

1) The user sends a query to each database n, to privately specify the required submodel W_θ given by,

$$Q_n = \begin{bmatrix} \frac{1}{f_1 - \alpha_n} \boldsymbol{e}_M(\theta) + \tilde{Z}_1 \\ \vdots \\ \frac{1}{f_\ell - \alpha_n} \boldsymbol{e}_M(\theta) + \tilde{Z}_\ell \end{bmatrix}, \quad n \in \{1, \dots, N\}, \quad (10)$$

where $\boldsymbol{e}_M(\theta)$ is the all zeros vector of size $M \times 1$ with a 1 at the θth position and $\tilde{Z}_i$ are random noise vectors.

2) In order to download the non-permuted version of the ith, $i \in \{1, \dots, |\tilde{V}|\}$, required subpacket (i.e., $V(i) = \tilde{P}(\tilde{V}(i))$) from the set $\tilde{V}$, database n picks the column $\tilde{V}(i)$ of the permutation-reversing matrix R_n given in (9) indicated by $R_n(:, \tilde{V}(i))$ and calculates the corresponding modified query given by,

$$Q_n^{[V(i)]} = \begin{bmatrix} R_n(1, \tilde{V}(i)) Q_n \\ \vdots \\ R_n(P, \tilde{V}(i)) Q_n \end{bmatrix} \quad (11)$$

$$= \begin{bmatrix} 1_{\{V(i)=1\}} \begin{bmatrix} \frac{1}{f_1 - \alpha_n} \boldsymbol{e}_M(\theta) \\ \vdots \\ \frac{1}{f_\ell - \alpha_n} \boldsymbol{e}_M(\theta) \end{bmatrix} + P_{\alpha_n}(\ell) \\ \vdots \\ 1_{\{V(i)=P\}} \begin{bmatrix} \frac{1}{f_1 - \alpha_n} \boldsymbol{e}_M(\theta) \\ \vdots \\ \frac{1}{f_\ell - \alpha_n} \boldsymbol{e}_M(\theta) \end{bmatrix} + P_{\alpha_n}(\ell) \end{bmatrix}, \quad (12)$$

[3] Note that all information on sparse subpackets (in both uplink and downlink) communicated between the users and the databases are in terms of the permuted indices of subpackets, so that the databases never learn the real subpacket indices or the underlying permutation.

where $P_{\alpha_n}(\ell)$ are noise vectors of size $M\ell\times 1$ consisting of polynomials in α_n of degree ℓ.

3) Then, the user downloads subpacket $V(i) = \tilde{P}(\tilde{V}(i))$, $i \in \{1,\ldots,|\tilde{V}|\}$ of the required submodel using the answers received by the N databases given by,

$$A_n^{[V(i)]} = S_n^T Q_n^{[V(i)]} \tag{13}$$

$$= \frac{1}{f_1-\alpha_n} W_{\theta,1}^{[V(i)]} + \ldots + \frac{1}{f_\ell-\alpha_n} W_{\theta,\ell}^{[V(i)]} + P_{\alpha_n}(3\ell+1), \tag{14}$$

from which the ℓ bits of subpacket $V(i)$, $i \in \{1,\ldots,|\tilde{V}|\}$ can be obtained from the N answers, given that $N = \ell + 3\ell + 2 = 4\ell + 2$ is satisfied. Thus, the subpacketization is $\ell = \frac{N-2}{4}$, and the reading cost is,

$$C_R = \frac{P\log_q P + |\tilde{V}|(N+\log_q P)}{L} = \frac{4r' + \frac{4}{N}(1+r')\log_q P}{1-\frac{2}{N}}, \tag{15}$$

where r', $0 \le r' \le 1$ is the sparsification rate in the downlink characterized by $|\tilde{V}| = P \times r'$.

C. Writing Phase at Time t

The writing phase of the PRUW scheme with sparsification consists of the following steps.

1) The user generates combined updates (one bit per subpacket) of the Pr non-zero subpackets and has zero as the combined update of the rest of the $P(1-r)$ subpackets. The update of subpacket s for database n is given by,[4]

$$U_n(s) = \begin{cases} 0, & s \in B^c, \\ \sum_{i=1}^{\ell} \tilde{\Delta}_{\theta,i}^{[s]} \prod_{j=1,j\neq i}^{\ell} (f_j - \alpha_n) & \\ \quad + \prod_{j=1}^{\ell}(f_j-\alpha_n)\hat{Z}_s, & s \in B, \end{cases} \tag{16}$$

where B is the set of (real) subpacket indices with non-zero updates, $\hat{Z}_s$ is a random noise bit and $\tilde{\Delta}_{\theta,i}^{[s]} = \frac{\Delta_{\theta,i}^{[s]}}{\prod_{j=1,j\neq i}^{\ell}(f_j-f_i)}$ with $\Delta_{\theta,i}^{[s]}$ being the update for the ith bit of subpacket s of W_θ.

2) The user permutes the updates of subpackets using $\tilde{P}$,

$$\hat{U}_n(i) = U_n(\tilde{P}(i)), \quad i = 1,\ldots,P. \tag{17}$$

3) Then, the user sends the following (update, position) pairs to each database n,

$$Y_n^{[j]} = (\hat{U}_n^{[j]}, k^{[j]}), \quad j = 1,\ldots,Pr, \tag{18}$$

where $k^{[j]}$ is the jth non-zero subpacket index (permuted) based on $\tilde{P}$ and $\hat{U}_n^{[j]}$ is the corresponding update.

4) Based on the received (update, position) pairs, each database constructs an update vector $\tilde{U}_n$ of size $P \times 1$ with $\hat{U}_n^{[j]}$ placed as the $k^{[j]}$th entry and zeros elsewhere,

$$\tilde{U}_n = \sum_{j=1}^{Pr} \hat{U}_n^{[j]} \boldsymbol{e}_P(k^{[j]}). \tag{19}$$

[4]A permuted version of these updates is sent to the databases.

5) $\tilde{U}_n$ in (19) contains the combined updates of the form (16) arranged in a random permutation given by $\tilde{P}$. The databases are unable to determine the true indices of all zero subpackets since $\tilde{P}$ is not known by the databases. However, for correctness in writing phase, the updates in $\tilde{U}_n$ must be rearranged in the correct order. This is done with the permutation-reversing matrix given in (9) as,

$$T_n = R_n\tilde{U}_n = R\tilde{U}_n + \prod_{i=1}^{\ell}(f_i-\alpha_n)P_{\alpha_n}(\ell), \tag{20}$$

where $P_{\alpha_n}(\ell)$ is a $P\times 1$ vector containing noise polynomials in α_n of degree ℓ, $R\tilde{U}_n$ contains all updates of all subpackets (including zeros) in the correct order, while $\prod_{i=1}^{\ell}(f_i-\alpha_n)P_{\alpha_n}(\ell)$ contains random noise, that hides the indices of the zero update subpackets.

6) The incremental update is calculated in the same way as described in [18] in each subpacket as,

$$h(s) = D_n \times T_n(s) \times Q_n \tag{21}$$

$$= D_n \times U_n(s) \times Q_n + D_n \times P_{\alpha_n}(2\ell) \tag{22}$$

$$= \begin{cases} \begin{bmatrix} \Delta_{\theta,1}^{[s]} \boldsymbol{e}_M(\theta) \\ \vdots \\ \Delta_{\theta,\ell}^{[s]} \boldsymbol{e}_M(\theta) \end{bmatrix} + \begin{bmatrix} (f_1-\alpha_n)P_{\alpha_n}(2\ell) \\ \vdots \\ (f_\ell-\alpha_n)P_{\alpha_n}(2\ell) \end{bmatrix}, & s \in B, \\ \begin{bmatrix} (f_1-\alpha_n)P_{\alpha_n}(2\ell) \\ \vdots \\ (f_\ell-\alpha_n)P_{\alpha_n}(2\ell) \end{bmatrix}, & s \in B^c \end{cases} \tag{23}$$

where $P_{\alpha_n}(2\ell)$ here are noise vectors of size $M\ell\times 1$ in (22) and $M\times 1$ in (23) with polynomials in α_n of degree 2ℓ and D_n is the scaling matrix given by,

$$D_n = \begin{bmatrix} (f_1-\alpha_n)I_M & \ldots & 0 \\ \vdots & \vdots & \vdots \\ 0 & \ldots & (f_\ell-\alpha_n)I_M \end{bmatrix}, \tag{24}$$

for $n \in \{1,\ldots,N\}$. $h(s)$ is in the same format as the storage and hence can be added to the existing storage to obtain the updated storage, i.e.,

$$S_n^{[t]}(s) = S_n^{[t-1]}(s) + h(s), \quad s = 1,\ldots,P. \tag{25}$$

The writing cost of the scheme is given by,

$$C_W = \frac{PrN(1+\log_q P)}{P\frac{N-2}{4}} = \frac{4r(1+\log_q P)}{1-\frac{2}{N}}. \tag{26}$$

Remark 2 *The union of permuted indices $k^{[j]}$ of the sparse subpackets in* (18) *sent by all users in the writing phase is what is described as $\tilde{V}$ in the reading phase of the next time instance in the example downlink sparsification protocol described in Section IV-B.*

Remark 3 *This problem can also be solved by considering a classical FSL setting (without sparsification) with P submod-*

els (i.e., $M = P$) and by using the private FSL scheme in [18] to update the sparse Pr submodels. However, in this case the cost of sending the queries Q_n given by $\frac{NM\ell}{L} = \frac{NP\ell}{L} = N$ is significantly large, and cannot be neglected.

Remark 4 *The proposed PRUW scheme with sparsification defined for private FSL can also be modified and applied to private FL. The reading and writing costs for this case are given by $C_R = \frac{3r'+\frac{3}{N}(1+r')\log_q P}{1-\frac{1}{N}}$ and $C_W = \frac{3r(1+\log_q P)}{1-\frac{1}{N}}$.*

D. Example

Assume that there are $P = 5$ subpackets in each submodel. The coordinator first picks a random permutation of $\{1, \ldots, 5\}$ out of the 5! options available. Let the realization of the permutation be $\tilde{P} = \{2, 5, 1, 3, 4\}$. The corresponding permutation-reversing matrix for database n, $n \in \{1, \ldots, N\}$, is given by,

$$R_n = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} + \prod_{i=1}^{\ell}(f_i - \alpha_n)\bar{Z}, \tag{27}$$

where $\bar{Z}$ is a random noise matrix of size 5×5 and ℓ is the subpacketization. The coordinator places matrix R_n at database n at the beginning of the process and sends $\tilde{P}$ to each user. Assume that a given user wants to update submodel θ at time t. Assuming the example downlink sparsification protocol described in Section IV-B, the user only needs to download subpackets that were updated at time $t-1$. Let the set of permuted indices of the subpackets updated by all users at time $t-1$ be $\tilde{V} = \{2, 3\}$, which is known by all databases; (see Remark 2). One designated database sends these permuted indices to each of the users at time t (who were also present at time $t-1$). Then, the user can obtain the real subpacket indices updated by all users at time $t-1$ using $V(i) = \tilde{P}(\tilde{V}(i))$ for $i = 1, 2$, i.e., $V = \{5, 1\}$. The user sends the query specifying the requirement of submodel θ given by (10) to database n.[5] Then, the databases send the parameters of the two subpackets in V as two bits, without learning the contents of V. To do this, each database first calculates the non-permuted query vector for each subpacket $V(i)$ using the permutation-reversing matrix and the query received. The query for subpacket $V(1) = 5$ is,

$$Q_n^{[5]} = \begin{bmatrix} R_n(1, \tilde{V}(1))Q_n \\ \vdots \\ R_n(P, \tilde{V}(1))Q_n \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ Q_n \end{bmatrix} + P_{\alpha_n}(\ell) \tag{28}$$

where $P_{\alpha_n}(\ell)$ is a vector of size $5M\ell \times 1$ consisting of polynomials in α_n of degree ℓ and $\mathbf{0}$ is the all zeros vector of size $M\ell \times 1$. Then, the answer from database n corresponding to subpacket $V(1) = 5$ is given by,

$$A_n^{[5]} = S_n^T Q_n^{[5]} \tag{29}$$

$$= \frac{1}{f_1 - \alpha_n} W_{\theta,1}^{[5]} + \ldots + \frac{1}{f_\ell - \alpha_n} W_{\theta,\ell}^{[5]} + P_{\alpha_n}(3\ell+1), \tag{30}$$

from which the ℓ bits of subpacket 5 of submodel θ can be correctly obtained if $N = 4\ell + 2$, which defines the subpacketization $\ell = \frac{N-2}{4}$. Similarly, the user can obtain subpacket 1 of W_θ by picking column $\tilde{V}(2) = 3$ of R_n in (28) and following the same process.

Once the user downloads and trains W_θ, the user generates the r fraction of subpackets with non-zero updates. Let the subpacket indices with non-zero updates be 1 and 4. The noisy updates generated by the user to be sent to database n according to (16) is given by $U_n = [U_n(1), 0, 0, U_n(4), 0]^T$ in the correct order. The user then permutes U_n based on the given permutation $\tilde{P}$, i.e., $\hat{U}_n(i) = U_n(\tilde{P}(i))$ for $i = \{1, \ldots, 5\}$,

$$\hat{U}_n = [0, 0, U_n(1), 0, U_n(4)]^T. \tag{31}$$

The user sends the values and the positions of the non-zero updates as $(U_n(1), 3)$ and $(U_n(4), 5)$ based on the permuted order. Each database receives these pairs and reconstructs (31),

$$\tilde{U}_n = U_n(1)\boldsymbol{e}_5(3) + U_n(4)\boldsymbol{e}_5(5) = \hat{U}_n. \tag{32}$$

To rearrange the updates back in the correct order privately, database n multiplies $\tilde{U}_n$ by the permutation-reversing matrix,

$$T_n = R_n \times \tilde{U}_n \tag{33}$$

$$= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tilde{U}_n + \prod_{i=1}^{\ell}(f_i - \alpha_n)\bar{Z} \times \tilde{U}_n \tag{34}$$

$$= [U_n(1), 0, 0, U_n(4), 0]^T + \prod_{i=1}^{\ell}(f_i - \alpha_n)P_{\alpha_n}(\ell), \tag{35}$$

since $U_n(1)$ and $U_n(4)$ are of the form $\sum_{i=1}^{\ell} \tilde{\Delta}_{\theta,i} \prod_{j=1, j\neq i}^{\ell}(f_j - \alpha_n) + \prod_{j=1}^{\ell}(f_j - \alpha_n)\hat{Z} = P_{\alpha_n}(\ell)$. The incremental update of subpacket s, is calculated by,

$$h(s) = D_n \times T_n(s) \times Q_n \tag{36}$$

$$= \begin{cases} \begin{bmatrix} \Delta_{1,1}^{[s]} \boldsymbol{e}_M(\theta) \\ \vdots \\ \Delta_{1,\ell}^{[s]} \boldsymbol{e}_M(\theta) \end{bmatrix} + \begin{bmatrix} (f_1 - \alpha_n)P_{\alpha_n}(2\ell) \\ \vdots \\ (f_\ell - \alpha_n)P_{\alpha_n}(2\ell) \end{bmatrix}, & s = 1, 4 \\ \begin{bmatrix} (f_1 - \alpha_n)P_{\alpha_n}(2\ell) \\ \vdots \\ (f_\ell - \alpha_n)P_{\alpha_n}(2\ell) \end{bmatrix}, & s = 2, 3, 5 \end{cases} \tag{37}$$

where $P_{\alpha_n}(2\ell)$ are vectors of size $M \times 1$ consisting of noise polynomials in α_n of degree 2ℓ. Since the incremental update is in the same format as the storage in (8), the existing storage can be updated as $S_n^{[t]}(s) = S_n^{[t-1]}(s) + h(s)$ for $s = 1, \ldots, 5$.

[5]Note that the query vector is of size $M\ell \times 1$ and is not considered in the cost calculation since $\frac{M\ell}{L}$ is negligible.

REFERENCES

[1] H. B. McMahan, E. Moore, et al. Communication efficient learning of deep networks from decentralized data. *AISTATS*, April 2017.

[2] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. *ACM Trans. on Intelligent Systems and Technology*, 10(2):1–19, January 2019.

[3] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1-2):1–210, June 2021.

[4] T. Li, A. K. Sahu, A. S. Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37:50–60, May 2020.

[5] J. Wangni, J. Wang, et al. Gradient sparsification for communication-efficient distributed optimization. In *NeurIPS*, December 2018.

[6] S. Li, Q. Qi, et al. GGS: General gradient sparsification for federated learning in edge computing. In *IEEE ICC*, June 2020.

[7] L. Barnes, H. Inan, B. Isik, and A. Ozgur. rTop-k: A statistical estimation approach to distributed SGD. *IEEE JSAIT*, 1(3):897–907, November 2020.

[8] P Han, S. Wang, and K. Leung. Adaptive gradient sparsification for efficient federated learning: An online learning approach. In *IEEE ICDCS*, November 2020.

[9] S. Shi, K. Zhao, Q. Wang, Z. Tang, and X. Chu. A convergence analysis of distributed SGD with communication-efficient gradient sparsification. In *IJCAI*, August 2019.

[10] Y. Sun, S. Zhou, Z. Niu, and D. Gunduz. Time-correlated sparsification for efficient over-the-air model aggregation in wireless federated learning. Available online at arXiv:2202.08420.

[11] E. Ozfatura, K. Ozfatura, and D. Gunduz. Time-correlated sparsification for communication-efficient federated learning. In *IEEE ISIT*, July 2021.

[12] D. Basu, D. Data, C. Karakus, and S. Diggavi. Qsparse-local-SGD: Distributed SGD with quantization, sparsification, and local computations. *IEEE JSAIT*, 1(1):217–226, May 2020.

[13] D. Alistarh, T. Hoefler, M. Johansson, S. Khirirat, N. Konstantinov, and C. Renggli. The convergence of sparsified gradient methods. In *NeurIPS*, December 2018.

[14] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *NeurIPS*, December 2017.

[15] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *AISTATS*, August 2020.

[16] N. Shlezinger, M. Chen, Y. Eldar, H. Poor, and S. Cui. Federated learning with quantization constraints. In *IEEE ICASSP*, May 2020.

[17] Z. Jia and S. A. Jafar. X-secure T-private federated submodel learning. In *IEEE ICC*, June 2021.

[18] S. Vithana and S. Ulukus. Efficient private federated submodel learning. In *IEEE ICC*, May 2022.

[19] Z. Jia and S. A. Jafar. X-secure T-private federated submodel learning with elastic dropout resilience. *IEEE Trans. on Info. Theory*, 68(8):5418–5439, August 2022.

[20] S. Vithana and S. Ulukus. Private read update write (PRUW) with storage constrained databases. In *IEEE ISIT*, June 2022.

[21] M. Kim and J. Lee. Information-theoretic privacy in federated submodel learning. *ICT express*, February 2022.

[22] C. Niu, F. Wu, S. Tang, L. Hua, R. Jia, C. Lv, Z. Wu, and G. Chen. Billion-scale federated learning on mobile clients: A submodel design with tunable privacy. In *MobiCom*, April 2020.

[23] C. Niu, F. Wu, S. Tang, L. Hua, R. Jia, C. Lv, Z. Wu, and G. Chen. Secure federated submodel learning. Available online at arXiv:1911.02254.

[24] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, November 1998.

[25] H. Sun and S. A. Jafar. The capacity of private information retrieval. *IEEE Trans. on Info. Theory*, 63(7):4075–4088, July 2017.

[26] K. Banawan and S. Ulukus. The capacity of private information retrieval from coded databases. *IEEE Trans. on Info. Theory*, 64(3):1945–1956, March 2018.

[27] C. Tian, H. Sun, and J. Chen. Capacity-achieving private information retrieval codes with optimal message size and upload cost. *IEEE Trans. on Info. Theory*, 65(11):7613–7627, November 2019.

[28] I. Samy, M. Attia, R. Tandon, and L. Lazos. Asymmetric leaky private information retrieval. *IEEE Trans. on Info. Theory*, 67(8):5352–5369, August 2021.

[29] H. Sun and S. A. Jafar. The capacity of symmetric private information retrieval. *IEEE Trans. on Info. Theory*, 65(1):329–322, January 2019.

[30] Z. Wang, K. Banawan, and S. Ulukus. Private set intersection: A multi-message symmetric private information retrieval perspective. *IEEE Trans. on Info. Theory*, 68(3):2001–2019, March 2022.

[31] H. Sun and S. A. Jafar. The capacity of robust private information retrieval with colluding databases. *IEEE Trans. on Info. Theory*, 64(4):2361–2370, April 2018.

[32] K. Banawan and S. Ulukus. Multi-message private information retrieval: Capacity results and near-optimal schemes. *IEEE Trans. on Info. Theory*, 64(10):6842–6862, October 2018.

[33] K. Banawan and S. Ulukus. The capacity of private information retrieval from Byzantine and colluding databases. *IEEE Trans. on Info. Theory*, 65(2):1206–1219, February 2019.

[34] S. Vithana, K. Banawan, and S. Ulukus. Semantic private information retrieval. *IEEE Trans. on Info. Theory*, 68(4):2635–2652, April 2022.

[35] S. Li and M. Gastpar. Single-server multi-message private information retrieval with side information: the general cases. In *IEEE ISIT*, June 2020.

[36] Z. Jia and S. A. Jafar. X-secure T-private information retrieval from MDS coded storage with Byzantine and unresponsive servers. *IEEE Trans. on Info. Theory*, 66(12):7427–7438, December 2020.

[37] S. Kadhe, B. Garcia, A. Heidarzadeh, S. El Rouayheb, and A. Sprintson. Private information retrieval with side information. *IEEE Trans. on Info. Theory*, 66(4):2032–2043, April 2020.

[38] S. Kumar, H.-Y. Lin, E. Rosnes, and A. G. i Amat. Achieving maximum distance separable private information retrieval capacity with linear codes. *IEEE Trans. on Info. Theory*, 65(7):4243–4273, July 2019.

[39] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE SSP*, May 2019.

[40] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *IEEE SSP*, May 2019.

[41] J. Geiping, H. Bauermeister, H. Droge, and M. Moeller. Inverting gradients–how easy is it to break privacy in federated learning? In *NeurIPS*, December 2020.

[42] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. In *NeurIPS*, December 2019.

[43] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE Infocom*, April-May 2019.

[44] S. Ulukus, S. Avestimehr, M. Gastpar, S. A. Jafar, R. Tandon, and C. Tian. Private retrieval, computing and learning: Recent progress and future challenges. *IEEE JSAC*, 40(3):729–748, March 2022.

[45] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *IEEE SSP*, May 2017.

[46] N. Carlini, C. Liu, U. Erlingsson, J. Kos, and D. Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *USENIX*, April 2019.

[47] K. Bonawitz, V. Ivanov, et al. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, October 2017.

[48] H. Ono and T. Takahashi. Locally private distributed reinforcement learning. Available online at arXiv:2001.11718.

[49] Y. Li, T. Chang, and C. Chi. Secure federated averaging algorithm with differential privacy. *IEEE MLSE*, September 2020.

[50] N. Agarwal, A. Suresh, F. Yu, S. Kumar, and H. B. McMahan. cpSGD: Communication-efficient and differentially-private distributed SGD. In *NeurIPS*, December 2018.

[51] B. Balle, G. Barthe, and M. Gaboardi. Privacy amplification by subsampling: Tight analyses via couplings and divergences. In *NeurIPS*, December 2018.

[52] S. Asoodeh and F. Calmon. Differentially private federated learning: An information-theoretic perspective. In *ICML-FL*, July 2020.

[53] U. Erlingsson, V. Feldman, et al. Amplification by shuffling: From local to central differential privacy via anonymity. In *ACM-SIAM symposium on discrete algorithms*, January 2019.

[54] B. Balle, J. Bell, A. Gascon, , and K. Nissim. The privacy blanket of the shuffle model. In *CRYPTO*, August 2019.

[55] A. Girgis, D. Data, et al. Shuffled model of differential privacy in federated learning. In *AISTAT*, April 2021.

[56] R. C. Geyer, T. Klein, and M. Nabi. Differentially private federated learning: A client level perspective. In *NeurIPS*, December 2017.