Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning

Jinhyun So, Başak Güler, *Member, IEEE*, and A. Salman Avestimehr, *Fellow, IEEE*

***Abstract*—Federated learning is a distributed framework for training machine learning models over the data residing at mobile devices, while protecting the privacy of individual users. A major bottleneck in scaling federated learning to a large number of users is the overhead of secure model aggregation across many users. In particular, the overhead of the state-of-the-art protocols for secure model aggregation grows quadratically with the number of users. In this article, we propose the first secure aggregation framework, named Turbo-Aggregate, that in a network with N users achieves a secure aggregation overhead of $O(N \log N)$, as opposed to $O(N^2)$, while tolerating up to a user dropout rate of 50%. Turbo-Aggregate employs a multi-group circular strategy for efficient model aggregation, and leverages additive secret sharing and novel coding techniques for injecting aggregation redundancy in order to handle user dropouts while guaranteeing user privacy. We experimentally demonstrate that Turbo-Aggregate achieves a total running time that grows almost linear in the number of users, and provides up to 40× speedup over the state-of-the-art protocols with up to $N = 200$ users. Our experiments also demonstrate the impact of model size and bandwidth on the performance of Turbo-Aggregate.**

***Index Terms*—Federated learning, privacy-preserving machine learning, secure aggregation.**

I. Introduction

FEDERATED learning is an emerging approach that enables model training over a large volume of decentralized data residing in mobile devices, while protecting the privacy of the individual users [1]–[4]. This is achieved by two key design principles. First, the training data is kept on the user device rather than sending it to a central server, and users locally perform model updates using their individual data. Second, local models are aggregated in a privacy-preserving framework, either at a central server (or in a distributed manner across the users) to update the global model. The global model is then pushed back to the mobile devices for inference. This process is demonstrated in Figure 1.

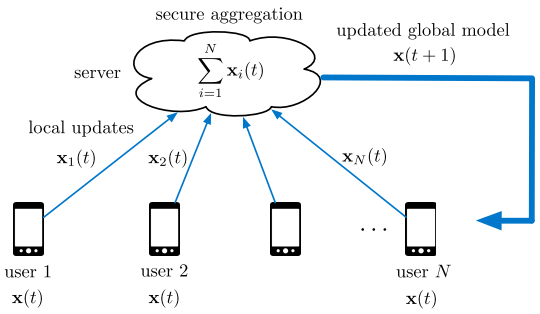


Fig. 1. Federated learning framework. At iteration t, the central server sends the current version of the global model, $\mathbf{x}(t)$, to the mobile users. User $i \in [N]$ updates the global model using its local data, and computes a local model $\mathbf{x}_i(t)$. The local models are then aggregated in a privacy-preserving manner. Using the aggregated models, the central server updates the global model $\mathbf{x}(t+1)$ for the next round, and pushes it back to the mobile users.

The privacy of individual models in federated learning is protected through what is known as a *secure aggregation* protocol [2], [3]. In this protocol, each user locally masks its own model using pairwise random masks and sends the masked model to the server. The pairwise masks have a unique property that once the masked models from all users are summed up at the server, the pairwise masks cancel out. As a result, the server learns the aggregate of all models, but no individual model is revealed to the server during the process. This is a key property for ensuring user privacy in secure federated learning. In contrast, conventional distributed training setups that do not employ secure aggregation may reveal extensive information about the private datasets of the users, which has been recently shown in [5]–[7]. To prevent such information leakage, secure aggregation protocols ensure that the individual update of each user is kept private, both from other users and the central server [2], [3]. A recent promising implementation of federated learning, as well as its application to Google keyboard query suggestions is demonstrated in [8]. Several other works have also demonstrated that leveraging the information that is distributed over many mobile users can increase the training performance dramatically, while ensuring data privacy and locality [9]–[11].

The overhead of secure model aggregation, however, creates a major bottleneck in scaling secure federated learning to a large number of users. More specifically, in a network with N users, the state-of-the-art protocols for secure aggregation require pairwise random masks to be generated between each

Manuscript received August 15, 2020; revised December 3, 2020 and January 21, 2021; accepted January 21, 2021. Date of publication January 26, 2021; date of current version March 16, 2021. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract HR001117C0053; in part by the Army Research Office under Award W911NF1810400; in part by NSF under Grant CCF-1703575 and Grant CCF-1763673; in part by the Office of Naval Research under Award N00014-16-1-2189; and in part by Intel. *(Corresponding author: Jinhyun So.)*

Jinhyun So and A. Salman Avestimehr are with the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: jinhyuns@usc.edu; avestimehr@ee.usc.edu).

Başak Güler is with the Department of Electrical and Computer Engineering, University of California at Riverside, Riverside, CA 92521 USA (e-mail: bguler@ece.ucr.edu).

Digital Object Identifier 10.1109/JSAIT.2021.3054610

pair of users (for hiding the local model updates), and therefore the overhead of secure aggregation grows quadratically in the number of users (i.e., $O(N^2)$) [2], [3]. This quadratic growth of secure aggregation overhead limits its practical applications to hundreds of users while the scale of current mobile systems is in the order of tens of millions [10].

Another key challenge in model aggregation is the dropout or unavailability of the users. Device availability and connection quality in mobile networks change rapidly, and users may drop from federated learning systems at any time due to various reasons, such as poor connectivity, making a phone call, low battery, etc. The design protocol hence needs to be robust to operate in such environments, where users can drop at any stage of the protocol execution. Furthermore, dropped or delayed users can lead to privacy breaches [3], and privacy guarantees should hold even in the case when users are dropped or delayed.

In this article, we introduce a novel secure aggregation framework for federated learning, named Turbo-Aggregate, with four salient features:

1) Turbo-Aggregate reduces the overhead of secure aggregation to $O(N \log N)$ from $O(N^2)$;
2) Turbo-Aggregate has provable robustness guarantees against up to a user dropout rate of 50%;
3) Turbo-Aggregate protects the privacy of the local model updates of each individual user, in the strong information-theoretic sense;
4) Turbo-Aggregate experimentally achieves a total running time that grows almost linear in the number of users, and provides up to $40\times$ speedup over the state-of-the-art with $N = 200$ users, in distributed implementation over Amazon EC2 cloud.

At a high level, Turbo-Aggregate is composed of three main components. First, Turbo-Aggregate employs a multi-group circular strategy for model aggregation. In particular, the users are partitioned into several groups, and at each aggregation stage, the users in one group pass the aggregated models of all the users in the previous groups and current group to users in the next group. We show that this structure enables the reduction of aggregation overhead to $O(N \log N)$ (from $O(N^2)$). However, there are two key challenges that need to be addressed in the proposed multi-group circular strategy for model aggregation. The first one is to protect the privacy of the individual user, i.e., the aggregation protocol should not allow the identification of individual model updates. The second one is handling the user dropouts. For instance, a user dropped at a higher group of the protocol may lead to the loss of the aggregated model information from all the previous groups, and collecting this information again from the lower groups may incur a large communication overhead.

The second key component is to leverage additive secret sharing [12], [13] to enable privacy and security of the users. In particular, additive sharing masks each local model by adding randomness in a way that can be cancelled out once the models are aggregated. Finally, the third component is to add aggregation redundancy via Lagrange coding [14] to enable robustness against delayed or dropped users. In particular, Turbo-Aggregate injects redundancy via Lagrange polynomial so that the added redundancy can be exploited to reconstruct the aggregated model amidst potential dropouts.

Turbo-Aggregate allows the use of both centralized and decentralized communication architectures. The centralized architecture refers to the communication model used in the conventional federated learning setup where all communication goes through a central server, i.e., the server acts as an access point [1], [3], [4]. The decentralized architecture, on the other hand, refers to the setup where mobile devices communicate directly with each other via an underlay communication network (e.g., a peer-to-peer network) [15], [16] without requiring a central server for secure model aggregation. Turbo-Aggregate also allows additional parallelization opportunities for communication, such as broadcasting and multi-casting.

We theoretically analyze the performance guarantees of Turbo-Aggregate in terms of the aggregation overhead, privacy protection, and robustness to dropped or delayed users. In particular, we show that Turbo-Aggregate achieves an aggregation overhead of $O(N \log N)$ and can tolerate a user dropout rate of 50%. We then quantify the privacy guarantees of our system. An important implication of dropped or delayed users is that they may lead to privacy breaches [2]. Accordingly, we show that the privacy-protection of our algorithm is preserved in such scenarios, i.e., when users are dropped or delayed.

We also provide extensive experiments to numerically evaluate the performance of Turbo-Aggregate. To do so, we implement Turbo-Aggregate for up to 200 users on the Amazon EC2 cloud, and compare its performance with the state-of-the-art secure aggregation protocol from [3]. We demonstrate that Turbo-Aggregate can achieve an overall execution time that grows almost linear in the number of users, and provides up to $40\times$ speedup over the state-of-the-art with 200 users. Furthermore, the overall execution time of Turbo-Aggregate remains stable as the user dropout rate increases, while for the benchmark protocol, the overall execution time significantly increases as the user dropout rate increases. We further study the impact of communication bandwidth on the performance of Turbo-Aggregate, by measuring the total running time with various bandwidth constraints. Our experimental results demonstrate that Turbo-Aggregate still provides substantial gain in environments with more severe bandwidth constraints.

II. Related Work

A potential solution for secure aggregation is to leverage cryptographic approaches, such as multiparty computation (MPC), homomorphic encryption, or differential privacy. MPC-based techniques mainly utilize Yao's garbled circuits or secret sharing (e.g., [17]–[20]). Their main bottleneck is the high communication cost, and communication-efficient implementations require an extensive offline computation part [19], [20]. A notable recent work is [21], which focuses on optimizing MPC protocols for network security and monitoring. Homomorphic encryption is a cryptographic secure computation scheme that allows aggregations to be performed on encrypted data [22]–[24]. However, the privacy guarantees of homomorphic encryption depends on the size of the encrypted data (more privacy requires a larger encrypted data

size), and performing computations in the encrypted domain is computationally expensive [25], [26]. Differential privacy is a noisy release mechanism that preserves the privacy of personally identifiable information, in that the removal of any single element from the dataset does not affect the computation outcomes significantly. As such, the computation outcomes cannot be used to infer much about any single individual element [27]. In the context of federated learning, differential privacy is mainly used to ensure that individual data points from the local datasets cannot be identified from the local updates sent to the server, by adding artificial noise to the local updates at the clients' side [9], [28], [29]. This approach entails a trade-off between convergence performance and privacy protection, i.e., stronger privacy guarantees lead to a degradation in the convergence performance. On the other hand, our focus is on ensuring that the server or a group of colluding users can learn nothing beyond the aggregate of all local updates, while preserving the accuracy of the model. This approach, also known as secure aggregation [2], [3], does not sacrifice the convergence performance.

A recent line of work has focused on secure aggregation by additive masking [3], [30]. In [30], users agree on pairwise secret keys using a Diffie-Hellman type key exchange protocol and then each user sends the server a masked version of their data, which contains the pairwise masks as well as an individual mask. The server can then sum up the masked data received from the users to obtain the aggregated value, as the summation of additive masks cancel out. If a user fails and drops out, the server asks the remaining users to send the sum of their pairwise keys with the dropped users added to their individual masks, and subtracts them from the aggregated value. The main limitation of this protocol is the communication overhead of this recovery phase, as it requires the entire sum of the missing masks to be sent to the server. Moreover, the protocol terminates if additional users drop during this phase.

A novel technique is proposed in [3] to ensure that the protocol is robust if additional users drop during the recovery phase. It also ensures that the additional information sent to the server does not breach privacy. To do so, the protocol utilizes pairwise random masks between users to hide the individual models. The cost of reconstructing these masks, which takes the majority of execution time, scales with respect to $O(N^2)$, with N corresponding to the number of users. The execution time of [3] increases as more users are dropped, as the protocol requires additional information corresponding to the dropped users. The recovery phase of our protocol does not require any additional information to be shared between the users, which is achieved by a coding technique applied to the additively secret shared data. Hence, the execution time of our algorithm stays almost the same as more and more users are dropped, the only overhead comes from the decoding phase whose contribution is very small compared to the overall communication cost.

Notable approaches to reduce the communication cost in federated learning include reducing the model size via quantization, or learning in a smaller parameter space [31]. In [32], a framework has been proposed for autotuning the parameters in secure federated learning, to achieve communication-efficiency. Another line of work has focused on approaches based on decentralized learning [33], [34] or edge-assisted hierarchical physical layer topologies [35]. Specifically, [35] utilizes edge servers to act as an intermediate aggregator for the local updates from edge devices. The global model is then computed at the central server by aggregating the intermediate computations available at the edge servers. These setups perform the aggregation using the clear (unmasked) model updates, i.e., the aggregation is not required to preserve the privacy of individual model updates. Our focus is different, as we study the secure aggregation problem which requires the server to learn no information about an individual update beyond the aggregated values. Finally, approaches that aim at alleviating the aggregation overhead by reducing the model size (e.g., quantization [31]) can also be leveraged in Turbo-Aggregate, which can be an interesting future direction.

Circular communication and training architectures have been considered previously in the context of distributed stochastic gradient descent on clear (unmasked) gradient updates, to reduce communication load [36] or to model data-heterogeneity [37]. Different from these setups, our key challenge in this work is handling user dropouts while ensuring user privacy, i.e., secure aggregation. Conventional federated learning frameworks consider a centralized communication architecture in which all communication between the mobile devices goes through a central server [1], [3], [4]. More recently, decentralized federated learning architectures without a central server have been considered for peer-to-peer learning on graph topologies [15] and in the context of social networks [16]. Model poisoning attacks on federated learning architectures have been analyzed in [38], [39]. Differentially-private federated learning frameworks have been studied in [28], [40]. A multi-task learning framework for federated learning has been proposed in [41], for learning several models simultaneously. References [42], [43] have explored federated learning frameworks to address fairness challenges and to avoid biasing the trained model towards certain users. Convergence properties of trained models are studied in [44].

III. System Model

In this section, we first discuss the basic federated learning model. Next, we introduce the secure aggregation protocol for federated learning and discuss the key parameters for performance evaluation. Finally, we present the state-of-the-art for secure aggregation.

A. Basic Federated Learning Model

Federated learning is a distributed learning framework that allows training machine learning models directly on the data held at distributed devices, such as mobile phones. The goal is to learn a single global model $\mathbf{x}$ with dimension d, using data that is generated, stored, and processed locally at millions of remote devices. This can be represented by minimizing a global objective function,

$$\min_{\mathbf{x}} L(\mathbf{x}) \text{ such that } L(\mathbf{x}) = \sum_{i=1}^{N} w_i L_i(\mathbf{x}), \tag{1}$$

where N is the total number of mobile users, L_i is the local objective function of user i, and $w_i \geq 0$ is a weight parameter assigned to user i to specify the relative impact of each user such that $\sum_i w_i = 1$. One natural setting of the weight parameter is $w_i = \frac{m_i}{m}$ where m_i is the number of samples of user i and $m = \sum_{i=1}^{N} m_i$.[1]

To solve (1), conventional federated learning architectures consider a centralized communication topology in which all communication between the individual devices goes through a central server [1], [3], [4], and no direct links are allowed between the mobile users. The learning setup is as demonstrated in Figure 1. At iteration t, the central server shares the current version of the global model, $\mathbf{x}(t)$, with the mobile users. Each user then updates the model using its local data. User $i \in [N]$ then computes a local model $\mathbf{x}_i(t)$. To increase communication efficiency, each user can update the local model over multiple local epochs before sending it to the server [1]. The local models of the N users are sent to the server and then aggregated by the server. Using the aggregated models, the server updates the global model $\mathbf{x}(t+1)$ for the next iteration. This update equation is given by

$$\mathbf{x}(t+1) = \sum_{i \in \mathcal{U}(t)} \mathbf{x}_i(t), \tag{2}$$

where $\mathcal{U}(t)$ denotes the set of participating users at iteration t. Then, the server pushes the updated global model $\mathbf{x}(t+1)$ to the mobile users.

B. Secure Aggregation Protocol for Federated Learning and Key Parameters

The basic federated learning model from Section III-A aims at addressing the privacy concerns over transmitting raw data to the server, by letting the training data remain on the user device and instead requiring only the local models to be sent to the server. However, as the local models still carry extensive information about the local datasets stored at the users, the server can reconstruct the private data from the local models by using a model inversion attack, which has been recently demonstrated in [5]–[7]. Secure aggregation has been introduced in [3] to address such privacy leakage from the local models. A secure aggregation protocol enables the computation of the aggregation operation in (2) while ensuring that the server learns no information about the local models $\mathbf{x}_i(t)$ beyond their aggregated value $\sum_{i=1}^{N} \mathbf{x}_i(t)$. In this article, our focus is on the aggregation phase in (2) and how to make this aggregation phase secure and efficient. In particular, our goal is to evaluate the aggregate of the local models

$$\mathbf{z} = \sum_{i \in \mathcal{U}} \mathbf{x}_i, \tag{3}$$

where we omit the iteration index t for simplicity. As we discuss in Section III-C and Appendix A in detail in the supplementary material, secure aggregation protocols build on cryptographic primitives that require all operations to be carried out over a finite field. Accordingly, similar to prior works [2], [3], we assume that the elements of $\mathbf{x}_i^{(l)}$ and $\mathbf{z}$ are from a finite field $\mathbb{F}_q$ for some field size q.

We evaluate the performance of a secure aggregation protocol for federated learning through the following key parameters.

1) *Robustness Guarantee:* We consider a network model in which each user can drop from the network with a probability $p \in [0, 1]$, called the user dropout rate. In a real world setting, the dropout rate varies between 0.06 and 0.1 [10]. The robustness guarantee quantifies the maximum user dropout rate that a protocol can tolerate with a probability approaching to 1 as $N \to \infty$ to correctly evaluate the aggregate of the surviving user models.
2) *Privacy Guarantee:* We consider a security model where the users and the server are honest but curious. We assume that up to T users can collude with each other as well as with the server for learning the models of other users. The privacy guarantee quantifies the maximum number of colluding entities that the protocol can tolerate for the individual user models to keep private.
3) *Aggregation Overhead:* The aggregation overhead, denoted by C, quantifies the asymptotic time complexity (i.e., runtime) with respect to the number of mobile users, N, for aggregating the models of all users in the network. Note that this includes both the computation and communication time complexities.

C. State-of-the-Art for Secure Aggregation

The state-of-the-art for secure aggregation in federated learning is the protocol proposed in [3]. In this protocol, each mobile user locally trains a model. By using pairwise random masking, the local models are securely aggregated through a central server, who then updates the global model. We present the details of the state-of-the-art in Appendix A in the supplementary material. This protocol achieves robustness guarantee to user dropout rate of up to $p = 0.5$, while providing privacy guarantee to up to $T = \frac{N}{2}$ colluding users. However, its aggregation overhead is quadratic with the number of users (i.e., $C = O(N^2)$). This quadratic aggregation overhead severely limits the network size for real-world applications [10].

Our goal in this article is to develop a secure aggregation protocol that can provide comparable robustness and privacy guarantees as the state-of-the-art, while achieving a significantly lower (almost linear) aggregation overhead.

IV. The Turbo-Aggregate Protocol

We now introduce the Turbo-Aggregate protocol for secure federated learning that can simultaneously achieve robustness guarantee to a user dropout rate of up to $p = 0.5$, privacy guarantee to up to $T = \frac{N}{2}$ colluding users, and aggregation overhead of $C = O(N \log N)$. Turbo-Aggregate is composed of three main components. First, it creates a multi-group circular aggregation structure for fast model aggregation. Second, it leverages additive secret sharing by adding randomness in a way that can be cancelled out once the models are aggregated, in order to guarantee the privacy of the users. Third, it adds aggregation redundancy via Lagrange polynomial in the model

[1]For simplicity, we assume that all users have equal-sized datasets i.e., a weight parameter assigned to user i satisfies $w_i = \frac{1}{N}$ for all $i \in [N]$.

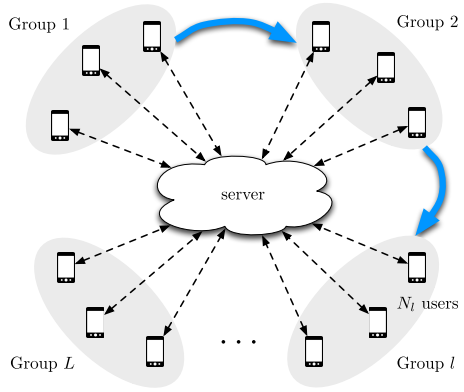


Fig. 2. Network topology with N users partitioned to L groups, with N_l users in group $l \in [L]$.

updates that are passed from one group to the next, so that the added redundancy can be exploited to reconstruct the aggregated model amidst potential user dropouts. We now describe each of these components in detail. An illustrative example is also presented in Appendix B in the supplementary material to demonstrate the execution of Turbo-Aggregate.

A. Multi-Group Circular Aggregation

Turbo-Aggregate computes the aggregate of the individual user models by utilizing a circular aggregation strategy. Given a mobile network with N users, this is done by first partitioning the users into L groups as shown in Figure 2, with N_l users in group $l \in [L]$, such that $\sum_{l\in[L]} N_l = N$. We consider a random partitioning strategy in which each user is assigned to one of the available groups uniformly at random, by using a bias-resistant public randomness generation protocol such as in [45]. We use $\mathcal{U}_l \subseteq [N_l]$ to represent the set of users that complete their part in the protocol (surviving users), and $\mathcal{D}_l = [N_l]\backslash\mathcal{U}_l$ to denote the set of dropped users.[2] We use $\mathbf{x}_i^{(l)}$ to denote the local model of user i in group $l \in [L]$, which is a vector of dimension d that corresponds to the parameters of their locally trained model. Then, we can rewrite (3) as

$$\mathbf{z} = \sum_{l\in[L]} \sum_{i\in\mathcal{U}_l} \mathbf{x}_i^{(l)}. \tag{4}$$

The elements of $\mathbf{x}_i^{(l)}$ and $\mathbf{z}$ are from a finite field $\mathbb{F}_q$ for some field size q. All operations are carried out over the finite field and we omit the modulo q operation for simplicity.

The dashed links in Figure 2 represent the communication links between the server and mobile users. In our general description, we assume that all communication takes place through a central server, via creating pairwise secure keys using a Diffie-Hellman type key exchange protocol [46] as in [3]. Turbo-Aggregate can also use decentralized communication architectures with direct links between devices, such as peer-to-peer communication, where users can communicate directly through an underlay communication network [15], [16]. Then, the aggregation steps are the same as the centralized setting except that messages are now communicated via direct links between the users, and a random election algorithm should be carried out to select one user (or multiple users, depending on the application) to aggregate the final sum at the final stage instead of the server. The detailed process of the final stage will be explained in Section IV-D.

Turbo-Aggregate consists of L execution stages performed sequentially. At stage $l \in [L]$, users in group l encode their inputs, including their trained models and the partial summation of the models from lower stages, and send them to users in group $l+1$. Next, users in group $l+1$ recover (decode) the missing information due to potentially dropped users, and then aggregate the received messages. At the end of the protocol, models of all surviving users will be aggregated.

The proposed coding and aggregation mechanism guarantees that no party (mobile users or the server) can learn an individual model, or a partial aggregate of a subset of models. The server learns nothing but the final aggregated model of all surviving users. This is achieved by leveraging additive secret sharing to mask the individual models, which we describe in the following.

B. Masking With Additive Secret Sharing

Turbo-Aggregate hides the individual user models using additive masks to protect their privacy against potential collusions between the interacting parties. This is done by a two-step procedure. In the first step, the server sends a random mask to each user, denoted by a random vector $\mathbf{u}_i^{(l)}$ for user $i \in [N_l]$ at group $l \in [L]$. Each user then masks its local model $\mathbf{x}_i^{(l)}$ as $\mathbf{x}_i^{(l)} + \mathbf{u}_i^{(l)}$. Since this random mask is known only by the server and the corresponding user, it protects the privacy of each user against potential collusions between any subset of the remaining users, as long as the server is honest. On the other hand, privacy may be breached if the server is adversarial and colludes with a subset of users. The second step of Turbo-Aggregate aims at protecting user privacy against such scenarios. In this second step, users generate additive secret sharing of the individual models for privacy protection against potential collusions between the server and the users. To do so, user i in group l sends a masked version of its local model to each user j in group $l+1$, given by

$$\widetilde{\mathbf{x}}_{i,j}^{(l)} = \mathbf{x}_i^{(l)} + \mathbf{u}_i^{(l)} + \mathbf{r}_{i,j}^{(l)}, \tag{5}$$

for $j \in [N_{l+1}]$, where $\mathbf{r}_{i,j}^{(l)}$ is a random vector such that $\sum_{j\in[N_{l+1}]} \mathbf{r}_{i,j}^{(l)} = 0$ for all $i \in [N_l]$. The role of additive secret sharing is not only to mask the model to provide privacy against collusions between the server and the users, but also to maintain the accuracy of aggregation by making the sum of the received data over the users in each group equal to the original data, as the vectors $\mathbf{r}_{i,j}^{(l)}$ cancel out.

In addition, each user holds a variable corresponding to the aggregated masked models from the previous group. For user i in group l, this variable is represented by $\widetilde{\mathbf{s}}_i^{(l)}$. At each stage of

[2]For modeling the user dropouts, we focus on the worst-case scenario, which is the case when a user drops during the execution of the corresponding group, i.e., when a user receives messages from the previous group but fails to propagate it to the next group.

Turbo-Aggregate, users in the active group update and propagate these variables to the next group. Aggregation of these masked models is defined via the recursive relation,

$$\widetilde{\mathbf{s}}_i^{(l)} = \frac{1}{N_{l-1}} \sum_{j\in[N_{l-1}]} \widetilde{\mathbf{s}}_j^{(l-1)} + \sum_{j\in\mathcal{U}_{l-1}} \widetilde{\mathbf{x}}_{j,i}^{(l-1)} \tag{6}$$

at user i in group $l > 1$, whereas the initial aggregation at group $l = 1$ is set as $\widetilde{\mathbf{s}}_i^{(1)} = \mathbf{0}$, for $i \in [N_1]$. While computing (6), any missing values in $\{\widetilde{\mathbf{s}}_j^{(l-1)}\}_{j\in[N_{l-1}]}$ (due to the users dropped in group $l-1$) is reconstructed via the recovery technique presented in Section IV-C.

User i in group l then sends the aggregated value in (6) to each user in group $l+1$. The average of the aggregated values from the users in group l consists of the models of the users up to group $l-1$, masked by the randomness sent from the server. This can be observed by defining the following partial summation, which can be computed by each user in group $l+1$,

$$\begin{aligned}\mathbf{s}^{(l+1)} &= \frac{1}{N_l} \sum_{i\in[N_l]} \widetilde{\mathbf{s}}_i^{(l)} \\ &= \frac{1}{N_{l-1}} \sum_{j\in[N_{l-1}]} \widetilde{\mathbf{s}}_j^{(l-1)} + \sum_{j\in\mathcal{U}_{l-1}} \mathbf{x}_j^{(l-1)} + \sum_{j\in\mathcal{U}_{l-1}} \mathbf{u}_j^{(l-1)} \quad (7)\\ &= \mathbf{s}^{(l)} + \sum_{j\in\mathcal{U}_{l-1}} \mathbf{x}_j^{(l-1)} + \sum_{j\in\mathcal{U}_{l-1}} \mathbf{u}_j^{(l-1)}, \quad (8)\end{aligned}$$

where (7) follows from $\sum_{j\in[N_{l-1}]} \mathbf{r}_{i,j}^{(l-1)} = 0$. With the initial partial summation $\mathbf{s}^{(2)} = \frac{1}{N_1}\sum_{i\in[N_1]} \widetilde{\mathbf{s}}_i^{(1)} = \mathbf{0}$, one can show that $\mathbf{s}^{(l+1)}$ is equal to the aggregation of the models of all surviving users in up to group $l-1$, masked by the randomness sent from the server,

$$\mathbf{s}^{(l+1)} = \sum_{m\in[l-1]} \sum_{j\in\mathcal{U}_m} \mathbf{x}_j^{(m)} + \sum_{m\in[l-1]} \sum_{j\in\mathcal{U}_m} \mathbf{u}_j^{(m)}. \tag{9}$$

At the final stage, the server obtains the final aggregate value from (9) and removes the random masks $\sum_{m\in[L]} \sum_{j\in\mathcal{U}_m} \mathbf{u}_j^{(m)}$. This approach works well if no user drops out during the execution of the protocol. On the other hand, if any user in group $l+1$ drops out, the random vectors masking the models of the l-th group in the summation (7) cannot be cancelled out. In the following, we propose a recovery technique that is robust to dropped or delayed users, based on coding theory principles.

C. Adding Redundancies to Recover the Data of Dropped or Delayed Users

The main intuition behind our recovery strategy is to encode the additive secret shares (masked models) in a way that guarantees secure aggregation when users are dropped or delayed. To do so, we leverage Lagrange coding [14], which has been applied to other problems such as offloading or collaborative machine learning in the privacy-preserving manner [47], [48]. The primary benefits of Lagrange coding over alternative codes that may also be used for introducing redundancy, such as other error-correcting codes, is that Lagrange coding enables us to perform the aggregation operation on the encoded models, and that the final result can be decoded from the computations performed on the encoded models. This is not necessarily true for other error-correcting codes, as they do not guarantee the recovery of the original computation results (i.e., the computations performed on the true values of the model parameters) from the computations performed on the encoded models. It encodes a given set of K vectors $(\mathbf{v}_1, \ldots, \mathbf{v}_K)$ by using a Lagrange interpolation polynomial. One can view this as embedding a given set of vectors on a Lagrange polynomial, such that each encoded value represents a point on the polynomial. The resulting encoding enables a set of users to compute a given polynomial function h on the encoded data in a way that any individual computation $\{h(\mathbf{v}_i)\}_{i\in[K]}$ can be reconstructed using any subset of $deg(h)(K-1)+1$ other computations. The reconstruction is done through polynomial interpolation. Therefore, one can reconstruct any missing value as long as a sufficient number of other computations are available, i.e., enough number of points are available to interpolate the polynomial. In our problem of gradient aggregation, the function of interest, h, would be linear and accordingly have degree 1, since it corresponds to the summation of all individual gradient vectors.

Turbo-Aggregate utilizes Lagrange coding for recovery against user dropouts, via a novel strategy that encodes the secret shared values to compute secure aggregation. More specifically, in Turbo-Aggregate, the encoding is performed as follows. Initially, user i in group l forms a Lagrange interpolation polynomial $f_i^{(l)} : \mathbb{F}_q \to \mathbb{F}_q^d$ of degree $N_{l+1}-1$ such that $f_i^{(l)}(\alpha_j^{(l+1)}) = \widetilde{\mathbf{x}}_{i,j}^{(l)}$ for $j \in [N_{l+1}]$, where $\alpha_j^{(l+1)}$ is an evaluation point allocated to user j in group $l+1$. This is accomplished by letting

$$f_i^{(l)}(z) = \sum_{j\in[N_{l+1}]} \widetilde{\mathbf{x}}_{i,j}^{(l)} \cdot \prod_{k\in[N_{l+1}]\setminus\{j\}} \frac{z-\alpha_k^{(l+1)}}{\alpha_j^{(l+1)}-\alpha_k^{(l+1)}}.$$

Then, another set of N_{l+1} distinct evaluation points $\{\beta_j^{(l+1)}\}_{j\in[N_{l+1}]}$ are allocated from $\mathbb{F}_q$ such that $\{\beta_j^{(l+1)}\}_{j\in[N_{l+1}]} \cap \{\alpha_j^{(l+1)}\}_{j\in[N_{l+1}]} = \varnothing$. Next, user $i \in [N_l]$ in group l generates the encoded model,

$$\bar{\mathbf{x}}_{i,j}^{(l)} = f_i^{(l)}\left(\beta_j^{(l+1)}\right), \tag{10}$$

and sends $\bar{\mathbf{x}}_{i,j}^{(l)}$ to user j in group $(l+1)$. In addition, user $i \in [N_l]$ in group l aggregates the encoded models $\{\bar{\mathbf{x}}_{j,i}^{(l-1)}\}_{j\in\mathcal{U}_{l-1}}$ received from the previous stage, with the partial summation $\mathbf{s}^{(l)}$ from (7) as

$$\bar{\mathbf{s}}_i^{(l)} = \mathbf{s}^{(l)} + \sum_{j\in\mathcal{U}_{l-1}} \bar{\mathbf{x}}_{j,i}^{(l-1)}. \tag{11}$$

The summation of the masked models in (6) and the summation of the coded models in (11) can be viewed as evaluations of a polynomial $g^{(l)}$ such that

$$\widetilde{\mathbf{s}}_i^{(l)} = g^{(l)}\left(\alpha_i^{(l)}\right), \tag{12}$$

$$\bar{\mathbf{s}}_i^{(l)} = g^{(l)}\left(\beta_i^{(l)}\right), \tag{13}$$

for $i \in [N_l]$, where $g^{(l)}(z) = \mathbf{s}^{(l)} + \sum_{j\in\mathcal{U}_{l-1}} f_j^{(l-1)}(z)$ is a polynomial function with degree at most $N_l - 1$. Then, user

$i \in [N_l]$ sends the set of messages $\{\widetilde{\mathbf{x}}_{i,j}^{(l)}, \bar{\mathbf{x}}_{i,j}^{(l)}, \widetilde{\mathbf{s}}_i^{(l)}, \bar{\mathbf{s}}_i^{(l)}\}$ to user j in group $l+1$.

Upon receiving the messages, user j in group $l+1$ reconstructs the missing terms in $\{\widetilde{\mathbf{s}}_i^{(l)}\}_{i\in[N_l]}$ (caused by the dropped users in group l), computes the partial sum $\mathbf{s}^{(l+1)}$ from (7), and updates the terms $\{\widetilde{\mathbf{s}}_j^{(l+1)}, \bar{\mathbf{s}}_j^{(l+1)}\}$ as in (6) and (11). Users in group $l+1$ can reconstruct each term in $\{\widetilde{\mathbf{s}}_i^{(l)}\}_{i\in[N_l]}$ as long as they receive at least N_l evaluations out of $2N_l$ evaluations from the users in group l. This is because $\{\widetilde{\mathbf{s}}_i^{(l)}, \bar{\mathbf{s}}_i^{(l)}\}_{i\in[N_l]}$ are evaluation points of the polynomial $g^{(l)}$ whose degree is at most $N_l - 1$. As a result, the model can be aggregated at each stage as long as at least half of the users at that stage are not dropped. As we will demonstrate in the proof of Theorem 1, as long as the drop rate of the users is below 50%, the fraction of dropped users at all stages will be below half with high probability, hence Turbo-Aggregate can proceed with model aggregation at each stage.

D. Final Aggregation and the Overall Turbo-Aggregate Protocol

For the final aggregation, we need a dummy stage to securely compute the aggregation of all user models, especially for the privacy of the local models of users in group L. To do so, we arbitrarily select a set of users who will receive and aggregate the models sent from the users in group L. They can be any surviving user who has participated in the protocol, and will be called user $j \in [N_{final}]$ in the final stage, where N_{final} is the number of users selected.

During this phase, users in group L mask their own model with additive secret sharing by using (5), generate the encoded data by using (10), and aggregate the models received from the users in group $(L-1)$ by using (6) and (11). Then, user i from group L sends $\{\widetilde{\mathbf{x}}_{i,j}^{(L)}, \bar{\mathbf{x}}_{i,j}^{(L)}, \widetilde{\mathbf{s}}_i^{(L)}, \bar{\mathbf{s}}_i^{(L)}\}$ to user j in the final stage.

Upon receiving the set of messages, user $j \in [N_{final}]$ in the final stage recovers the missing terms in $\{\widetilde{\mathbf{s}}_i^{(L)}\}_{i\in[N_L]}$, and aggregates them with the masked models,

$$\widetilde{\mathbf{s}}_j^{(final)} = \frac{1}{N_L}\sum_{i\in[N_L]} \widetilde{\mathbf{s}}_i^{(L)} + \sum_{i\in\mathcal{U}_L} \widetilde{\mathbf{x}}_{i,j}^{(L)}, \tag{14}$$

$$\bar{\mathbf{s}}_j^{(final)} = \frac{1}{N_L}\sum_{i\in[N_L]} \widetilde{\mathbf{s}}_i^{(L)} + \sum_{i\in\mathcal{U}_L} \bar{\mathbf{x}}_{i,j}^{(L)}, \tag{15}$$

and sends the resulting $\{\widetilde{\mathbf{s}}_j^{(final)}, \bar{\mathbf{s}}_j^{(final)}\}$ to the server.

The server then recovers the summations $\{\widetilde{\mathbf{s}}_j^{(final)}\}_{j\in[N_{final}]}$, by reconstructing any missing terms in (14) using the set of received values (14) and (15). Finally, the server computes the average of the summations from (14) and removes the random masks $\sum_{m\in[L]}\sum_{j\in\mathcal{U}_m}\mathbf{u}_j^{(m)}$ from the aggregate, which, as can be observed from (7)-(9), is equal to the aggregate of the individual models of all surviving users,

$$\frac{1}{N_{final}}\sum_{j\in[N_{final}]}\widetilde{\mathbf{s}}_j^{(final)} - \sum_{m\in[L]}\sum_{j\in\mathcal{U}_m}\mathbf{u}_j^{(m)} = \sum_{m\in[L]}\sum_{j\in\mathcal{U}_m}\mathbf{x}_j^{(m)}. \tag{16}$$

Having all above steps, the overall Turbo-Aggregate protocol is presented in Algorithm 1.

Algorithm 1 Turbo-Aggregate

input Local models $\mathbf{x}_i^{(l)}$ of users $i \in [N_l]$ in group $l \in [L]$.
output Aggregated model $\sum_{l\in[L]}\sum_{i\in\mathcal{U}_l}\mathbf{x}_i^{(l)}$.

for group $l = 1, \ldots, L$ **do**
  for user $i = 1, \ldots, N_l$ **do**
    Compute the masked model $\{\widetilde{\mathbf{x}}_{i,j}^{(l)}\}_{l\in[N_{l+1}]}$ from (5).
    Generate the encoded model $\{\bar{\mathbf{x}}_{i,j}^{(l)}\}_{j\in[N_{l+1}]}$ from (10).
    if $l = 1$ **then**
      Initialize $\widetilde{\mathbf{s}}_i^{(1)} = \bar{\mathbf{s}}_i^{(1)} = \mathbf{0}$.
    else
      Reconstruct the missing values in $\{\widetilde{\mathbf{s}}_k^{(l-1)}\}_{k\in[N_{l-1}]}$ due to the dropped users in group $l-1$.
      Update the aggregate value $\widetilde{\mathbf{s}}_i^{(l)}$ from (6).
      Compute the coded aggregate value $\bar{\mathbf{s}}_i^{(l)}$ from (11).
    Send $\{\widetilde{\mathbf{x}}_{i,j}^{(l)}, \bar{\mathbf{x}}_{i,j}^{(l)}, \widetilde{\mathbf{s}}_i^{(l)}, \bar{\mathbf{s}}_i^{(l)}\}$ to user $j \in [N_{l+1}]$ in group $l+1$ ($j \in [N_{final}]$ if $l = L$).
for user $i = 1, \ldots, N_{final}$ **do**
  Reconstruct the missing values in $\{\widetilde{\mathbf{s}}_k^{(L)}\}_{k\in[N_L]}$ due to the dropped users in group L.
  Compute $\widetilde{\mathbf{s}}_i^{(final)}$ from (14) and $\bar{\mathbf{s}}_i^{(final)}$ from (15).
  Send $\{\widetilde{\mathbf{s}}_i^{(final)}, \bar{\mathbf{s}}_i^{(final)}\}$ to the server.
Server computes the final aggregated model from (16).

V. Theoretical Guarantees of Turbo-Aggregate

In this section, we formally state our main theoretical result.

Theorem 1: Turbo-Aggregate can simultaneously achieve:

1) robustness guarantee to any user dropout rate $p < 0.5$, with probability approaching to 1 as the number of users $N \to \infty$,
2) privacy guarantee against up to $T = (0.5-\epsilon)N$ colluding users, with probability approaching to 1 as the number of users $N \to \infty$, and for any $\epsilon > 0$,
3) aggregation overhead of $C = O(N \log N)$.

Remark 1: Theorem 1 states that Turbo-Aggregate can tolerate up to 50% user dropout rate and $\frac{N}{2}$ collusions between the users, simultaneously. Turbo-Aggregate can guarantee robustness against an even higher number of user dropouts by sacrificing the privacy guarantee as a trade-off. Specifically, when we generate and communicate k set of evaluation points during Lagrange coding, we can recover the partial aggregations by decoding the polynomial in (12) as long as each user receives N_l evaluations, i.e., $(1+k)(N_l - pN_l) \geq N_l$. As a result, Turbo-Aggregate can tolerate up to a $p < \frac{k}{1+k}$ user dropout rate. On the other hand, the individual models will be revealed whenever $T(k+1) \geq N$. In this case, one can guarantee privacy against up to $(\frac{1}{k+1} - \epsilon)N$ colluding users for any $\epsilon > 0$. This demonstrates a trade-off between robustness and privacy guarantees achieved by Turbo-Aggregate, that is, one can increase the robustness guarantee by reducing the privacy guarantee and vice versa.

Proof: The proof of Theorem 1 is presented in Appendix C in the supplementary material. ∎

As we showed in the proof of Theorem 1, Turbo-Aggregate achieves its robustness and privacy guarantees by choosing a group size of $N_l = \frac{1}{c}\log N$ for all $l \in [L]$ where

$c \triangleq \min\{D(0.5||p), D(0.5||\frac{T}{N})\}$ and $D(a||b)$ is the Kullback-Leibler (KL) distance between two Bernoulli distributions with parameter a and b [49]. We can further reduce the aggregation overhead if we choose a smaller group size N_l. However, we cannot further reduce the group size beyond $O(\log N)$ because when $0 < D(0.5||p) < 1$ ($\frac{1}{c} > 1$) and $N_l = \log N$, the probability that Turbo-Aggregate guarantees the accuracy of full model aggregation goes to 0 with sufficiently large number of users, which is stated in Theorem 2.

Theorem 2 (Converse): When $0 < D(0.5||p) < 1$ and $N_l = \log N$ for all $l \in [L]$, the probability that Turbo-Aggregate achieves the robustness guarantee to any user dropout rate $p < 0.5$ goes to 0 as the number of users $N \to \infty$.

Proof: The proof of Theorem 2 is presented in Appendix D in the supplementary material. ■

A. Generalized Turbo-Aggregate

Theorem 1 states that the privacy of each *individual model* is guaranteed against any collusion between the server and up to $\frac{N}{2}$ users. On the other hand, a collusion between the server and a subset of users can reveal the *partial aggregation* of a group of honest users. For instance, a collusion between the server and a user in group l can reveal the partial aggregation of the models of all users up to group $l-2$, as the colluding server can remove the random masks in (9). However, the privacy protection can be strengthened to guarantee the privacy of *any* partial aggregation, i.e., the aggregate of any subset of user models, with a simple modification.

The modified protocol follows the same steps in Algorithm 1 except that the random mask $\mathbf{u}_i^{(l)}$ in (5) is generated by each user individually, instead of being generated by the server. At the end of the aggregation phase, the server learns $\sum_{m\in[L]}\sum_{j\in\mathcal{U}_m}(\mathbf{x}_j^{(m)} + \mathbf{u}_j^{(m)})$. Simultaneously, the protocol executes an additional random partitioning strategy to aggregate the random masks $\mathbf{u}_j^{(m)}$, at the end of which the server obtains $\sum_{m\in[L]}\sum_{j\in\mathcal{U}_m}\mathbf{u}_j^{(m)}$ and recovers $\sum_{m\in[L]}\sum_{j\in\mathcal{U}_m}\mathbf{x}_j^{(m)}$. In this second partitioning, N users are randomly allocated into L groups with a group size of N_l. User i in group $l' \in [L]$ then secret shares $\mathbf{u}_i^{(l')}$ with the users in group $l'+1$, by generating and sending a secret share denoted by $[\mathbf{u}_i^{(l')}]_j$ to user j in group $l'+1$. For secret sharing, we utilize Shamir's $\frac{N_l}{2}$-out-of-N_l secret sharing protocol [18]. Let $\mathcal{U}_{l'}^{'}$ denote the surviving users in group l' in the second partitioning. User i in group l' then aggregates the received secret shares $\sum_{j\in\mathcal{U}_{l'-1}^{'}}[\mathbf{u}_j^{(l'-1)}]_i$, which in turn is a secret share of $\sum_{j\in\mathcal{U}_{l'-1}^{'}}\mathbf{u}_j^{(l'-1)}$, and sends the sum to the server. Finally, the server reconstructs $\sum_{j\in\mathcal{U}_{l'}^{'}}\mathbf{u}_j^{(l')}$ for all $l' \in [L]$ and recovers the aggregate of the individual models of all surviving users by subtracting $\{\sum_{j\in\mathcal{U}_{l'}^{'}}\mathbf{u}_j^{(l')}\}_{l'\in[L]}$ from the aggregate $\sum_{m\in[L]}\sum_{j\in\mathcal{U}_m}(\mathbf{x}_j^{(m)} + \mathbf{u}_j^{(m)})$.

In this generalized version of Turbo-Aggregate, the privacy of any partial aggregation, i.e., the aggregate of any subset of user models, can be protected as long as a collusion between the server and the users does not reveal the aggregation of the random masks, $\sum_{j\in\mathcal{U}_l}\mathbf{u}_j^{(l)}$ in (9) for any $l \in [L]$. Since there are at least $\frac{N}{2}$ unknown random masks generated by honest users and the server only knows $L = \frac{N}{N_l}$ equations, i.e., $\{\sum_{j\in\mathcal{U}_{l'}^{'}}\mathbf{u}_j^{(l)}\}_{l\in[L]}$, the server cannot calculate $\sum_{j\in\mathcal{U}_l}\mathbf{u}_j^{(l)}$ for any $l \in [L]$. Therefore, a collusion between the server and users cannot reveal the partial aggregate as they cannot remove the random masks in (9). We now formally state the privacy guarantee, robustness guarantee, and aggregation overhead of the generalized Turbo-Aggregate protocol in Theorem 3.

Theorem 3: Generalized Turbo-Aggregate simultaneously achieves 1), 2), and 3) from Theorem 1. In addition, it provides privacy guarantee for the partial aggregate of any subset of user models, against any collusion between the server and up to $T = (0.5-\epsilon)N$ users for any $\epsilon > 0$, with probability approaching to 1 as the number of users $N \to \infty$.

Proof: The proof of Theorem 3 is presented in Appendix E in the supplementary material. ■

TABLE I
SUMMARY OF SIMULATION PARAMETERS

Variable	Definition	Value
N	number of users	4 ~ 200
d	model size (32 bit entries)	100000
p	dropout rate	10%, 30%, 50%
q	field size	$2^{32}-5$
	maximum bandwidth constraint	100Mbps ~ 1Gbps

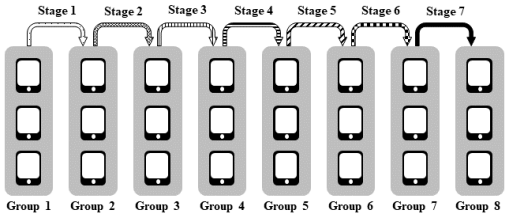
VI. EXPERIMENTS

In this section, we evaluate the performance of Turbo-Aggregate by experiments over up to $N = 200$ users for various user dropout rates and bandwidth conditions.

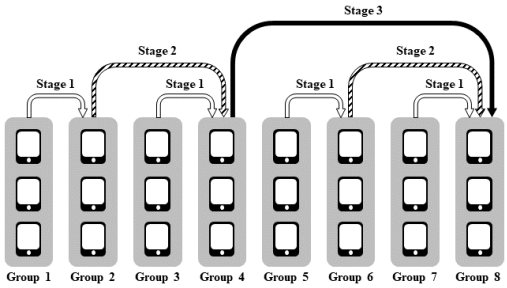
A. Experiment Setup

Platform: In our experiments, we implement Turbo-Aggregate on a distributed platform by using `FedML` library [50], and examine its total running time with respect to the state-of-the-art [3]. Computation is performed in a distributed network over the Amazon EC2 cloud using `m3.medium` machine instances. Communication is implemented using the `MPI4Py` [51] message passing interface on `Python`. The default setting for the maximum bandwidth constraint of `m3.medium` machine instances is 1*Gbps*. The model size, d, is fixed to 100,000 with 32 bit entries, and the field size, q, is set as the largest prime within 32 bits. We summarize the simulation parameters in Table I.

Modeling User Dropouts: To model the dropped users in Turbo-Aggregate, we randomly select pN_l users out of N_l users in group $l \in [L]$ where p is the dropout rate. We consider the worst case scenario where the selected users drop after receiving the messages sent from the previous group (users in group $l-1$) and do not send their messages to users in group $l+1$. To model the dropped users in the benchmark protocol, we follow the scenario in [3]. We randomly select pN users out of N users, which artificially drop after sending their masked models. In this case, the server has to reconstruct



(a) Turbo-Aggregate. Each arrow is carried out sequentially. Turbo-Aggregate requires 7 stages.



(b) Turbo-Aggregate+. Arrows for the same execution stage are carried out simultaneously. Turbo-Aggregate+ requires only 3 stages

Fig. 3. Example networks with $N = 24$, $N_l = 3$ and $L = 8$. An arrow represents that users in one group generate and send messages to the users in the next group.

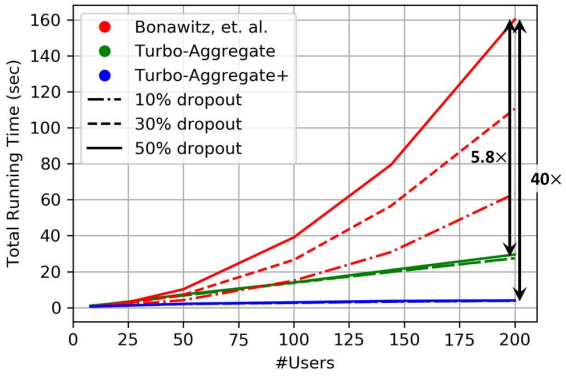


Fig. 4. Total running time of Turbo-Aggregate versus the benchmark protocol [3] as the number of users increases, for various user dropout rates.

the pairwise seeds of the dropped users and execute a pseudo random generator using the reconstructed seeds to remove the random masks (the details are provided in Appendix A in the supplementary material).

Implemented Schemes: We implement the following schemes for performance evaluation. For the schemes with Turbo-Aggregate, we use $N_l = \log N$.

1) *Turbo-Aggregate:* For our first implementation, we directly implement Turbo-Aggregate as described in Section IV, where the L execution stages are performed sequentially.
2) *Turbo-Aggregate+:* We can speed up Turbo-Aggregate by parallelizing the L execution stages. To do so, we again utilize the circular aggregation topology but leverage a tree structure for flooding the information between different groups across the network, which reduces the required number of execution stages from $L-1$ to $\log L$. We refer to this implementation as Turbo-Aggregate+. Figure 3 demonstrates the difference between Turbo-Aggregate+ and Turbo-Aggregate through an example network of $N = 24$ users and $L = 8$ groups. Turbo-Aggregate+ requires only 3 stages to complete the protocol while Turbo-Aggregate carries out each execution stage sequentially and requires 7 stages.
3) *Benchmark:* We implement the benchmark protocol [3] where a server mediates the communication between users to exchange the information required for key agreements (rounds of advertising and sharing keys) and users send their masked models to the server (masked input collection). One can also speed up the rounds of advertising and sharing keys by allowing users to communicate in parallel. However, this has minimal effect on the total running time of the protocol, as the total running time is dominated by the overhead when the server generates the pairwise masks [3].

B. Performance Evaluation

For performance analysis, we measure the total running time for a single round of secure aggregation with each protocol while increasing the number of users N gradually for different user dropout rates. We use synthesized vectors for locally trained models and do not include the local training time in the total running time. One can also consider the entire learning process and since all other steps remain the same for the three schemes, we expect the same speedup in the aggregation phase. Our results are demonstrated in Figure 4. We make the following key observations.

- Total running time of Turbo-Aggregate and Turbo-Aggregate+ are almost linear in the number of users, while for the benchmark protocol, the total running time is quadratic in the number of users.
- Turbo-Aggregate and Turbo-Aggregate+ provide a stable total running time as the user dropout rate increases. This is because the encoding and decoding time of Turbo-Aggregate do not change significantly when the dropout rate increases, and we do not require additional information to be transmitted from the remaining users when some users are dropped or delayed. On the other hand, for the benchmark protocol, the running time significantly increases as the dropout rate increases. This is because the total running time is dominated by the reconstruction of pairwise masks at the server, which substantially increases as the number of dropped users increases.
- Turbo-Aggregate and Turbo-Aggregate+ provide a speedup of up to 5.8× and 40× over the benchmark, respectively, for a user dropout rate of up to 50% with $N = 200$ users. This gain is expected to increase further as the number of users increases.

To illustrate the impact of user dropouts, we present the breakdown of the total running time of the three schemes and the corresponding observations in Appendix F1 in the supplementary material. We further study the impact of the bandwidth, by measuring the total running time with various

communication bandwidth constraints. Turbo-Aggregate provides substantial gain over the state-of-the-art in environments with more severe bandwidth constraints. The details of these additional experiments are presented in Appendix F2 in the supplementary material.

In this section, we have primarily focused on the aggregation phase and measured a single round of the secure aggregation phase with synthesized vectors for the locally trained models. This is due to the fact that these vectors can be replaced with any trained model using the real world federated learning setups. We further investigate the performance of Turbo-Aggregate in real world federated learning setups by implementing both training phase and aggregation phase. Turbo-Aggregate still provides substantial speedup over the benchmark, which is detailed in Appendix F3 in the supplementary material.

VII. Conclusion

This article presents the first secure aggregation framework that theoretically achieves an aggregation overhead of $O(N \log N)$ in a network with N users, as opposed to the prior $O(N^2)$ overhead, while tolerating up to a user dropout rate of 50%. Furthermore, via experiments over Amazon EC2, we demonstrated that Turbo-Aggregate achieves a total running time that grows almost linearly in the number of users, and provides up to $40\times$ speedup over the state-of-the-art scheme with $N = 200$ users.

Turbo-Aggregate is particularly suitable for wireless topologies, in which network conditions and user availability can vary rapidly, as Turbo-Aggregate can provide a resilient framework to handle such unreliable network conditions. Specifically, if some users cause unexpected delays due to unstable connection, Turbo-Aggregate can simply treat them as user dropouts and can reconstruct the information of dropped or delayed users in the previous groups as long as half of the users remain. One may also leverage the geographic heterogeneity of wireless networks to better form the communication groups in Turbo-Aggregate. An interesting future direction would be to explore how to optimize the multi-group communication structure of Turbo-Aggregate based on the specific topology of the users, as well as the network conditions.

In this work, we have focused on protecting the privacy of individual models against an honest-but-curious server and up to T colluding users so that no information is revealed about the individual models beyond their aggregated value. If one would like to further limit the information that may be revealed from the aggregated model, differential privacy can be utilized to ensure that the individual data points cannot be identified from the aggregated model. All the benefits of differential privacy could be applied to our approach by adding noise to the local models before the aggregation phase in Turbo-Aggregate. Combining these two techniques is another interesting future direction.

Finally, the implementation of Turbo-Aggregate in a real-world large-scale distributed system would be another interesting future direction. This would require addressing the following three challenges. First, the computation complexity of implementing the random grouping strategy may increase as the number of users increases. Second, Turbo-Aggregate currently focuses on protecting the privacy against honest-but-curious adversaries. In settings with malicious (Byzantine) adversaries who wish to manipulate the global model by poisoning their local datasets, one may require additional strategies to protect the resilience of the trained model. One approach is combining secure aggregation with an outlier detection algorithm as proposed in [52], which has a communication cost of $O(N^2)$ that limits its scalability to large federated learning systems. It would be an interesting direction to leverage Turbo-Aggregate to address this challenge, i.e., develop a communication-efficient secure aggregation strategy against Byzantine adversaries. Third, communication may still be a bottleneck in severely resource-constrained systems since users need to exchange the masked models with each other, whose size is as large as the size of the global model. To overcome this bottleneck, one may leverage model compression techniques or group knowledge transfer [53].

Acknowledgment

The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Int. Stat. (AISTATS)*, 2017, pp. 1273–1282.
[2] K. Bonawitz *et al.*, "Practical secure aggregation for federated learning on user-held data," in *Proc. Conf. Neural Inf. Process. Syst.*, 2016, p. 5.
[3] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 1175–1191.
[4] P. Kairouz *et al.*, "Advances and open problems in federated learning," 2019. [Online]. Available: arXiv:1912.04977.
[5] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 14774–14784.
[6] 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 *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2019, pp. 2512–2520.
[7] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients—How easy is it to break privacy in federated learning?" 2020. [Online]. Available: arXiv:2003.14053.
[8] T. Yang *et al.*, "Applied federated learning: Improving google keyboard query suggestions," 2018. [Online]. Available: arXiv:1812.02903.
[9] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–14.
[10] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," in *Proc. 2nd SysML Conf.*, 2019.
[11] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020.
[12] A. Beimel, "Secret-sharing schemes: A survey," in *Proc. Int. Conf. Coding Cryptol.*, 2011, pp. 11–46.
[13] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Found. Trends Privacy Security*, vol. 2, nos. 2–3, pp. 70–246, 2018.
[14] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," in *Proc. Int. Conf. Artif. Int. Stat. (AISTATS)*, 2019, pp. 1215–1225.
[15] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar, "Peer-to-peer federated learning on graphs," 2019. [Online]. Available: arXiv:1901.11173.
[16] C. He, C. Tan, H. Tang, S. Qiu, and J. Liu, "Central server free federated learning over single-sided trust social networks," 2019. [Online]. Available: arXiv:1910.04956.

[17] A. C. Yao, "Protocols for secure computations," in *Proc. IEEE Annu. Symp. Found. Comput. Sci.*, 1982, pp. 160–164.
[18] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[19] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proc. ACM Symp. Theory Comput.*, 1988, pp. 1–10.
[20] Z. Beerliová-Trubíniová and M. Hirt, "Perfectly-secure MPC with linear communication complexity," in *Proc. Theory Cryptogr. Conf.*, 2008, pp. 213–230.
[21] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics," *Network*, vol. 1, 2010, Art. no. 101101.
[22] I. Leontiadis, K. Elkhiyaoui, and R. Molva, "Private and dynamic time-series data aggregation with trust relaxation," in *Proc. Int. Conf. Cryptol. Netw. Security*, 2014, pp. 305–320.
[23] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2010, pp. 735–746.
[24] S. Halevi, Y. Lindell, and B. Pinkas, "Secure computation on the Web: Computing without simultaneous interaction," in *Proc. Annu. Cryptol. Conf.*, 2011, pp. 132–150.
[25] C. Gentry and D. Boneh, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009.
[26] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proc. Annu. Cryptol. Conf.*, 2012, pp. 643–662.
[27] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proc. Theory Cryptol. Conf.*, 2006, pp. 265–284.
[28] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," 2017. [Online]. Available: arXiv:1712.07557.
[29] K. Wei *et al.*, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, 2020.
[30] G. Ács and C. Castelluccia, "I have a dream! (differentially private smart metering)," in *Proc. Int. Workshop Inf. Hiding*, 2011, pp. 118–132.
[31] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *Proc. Conf. Neural Inf. Process. Syst. Workshop Private Multi-Party Mach. Learn.*, 2016.
[32] K. Bonawitz, F. Salehi, J. Konečnỳ, B. McMahan, and M. Gruteser, "Federated learning with autotuned communication-efficient secure aggregation," 2019. [Online]. Available: arXiv:1912.00131.
[33] L. He, A. Bian, and M. Jaggi, "COLA: Decentralized linear learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 4536–4546.
[34] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5330–5340.
[35] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Edge-assisted hierarchical federated learning with non-iid data," 2019. [Online]. Available: arXiv:1905.06641.
[36] Y. Li, M. Yu, S. Li, S. Avestimehr, N. S. Kim, and A. Schwing, "Pipe-SGD: A decentralized pipelined sgd framework for distributed deep net training," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8045–8056.
[37] H. Eichner, T. Koren, H. B. McMahan, N. Srebro, and K. Talwar, "Semi-cyclic stochastic gradient descent," 2019. [Online]. Available: arXiv:1904.10120.
[38] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," 2018. [Online]. Available: arXiv:1811.12470.
[39] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," 2019. [Online]. Available: arXiv:1911.11815.
[40] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, "Can you really backdoor federated learning?" 2019. [Online]. Available: arXiv:1911.07963.
[41] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4424–4434.
[42] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," 2019. [Online]. Available: arXiv:1902.00146.
[43] T. Li, M. Sanjabi, and V. Smith, "Fair resource allocation in federated learning," 2019. [Online]. Available: arXiv:1905.10497.
[44] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-IID data," 2019. [Online]. Available: arXiv:1907.02189.
[45] E. Syta *et al.*, "Scalable bias-resistant distributed randomness," in *Proc. IEEE Symp. Security Privacy (SP)*, 2017, pp. 444–460.
[46] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, Sep. 2006.
[47] J. So, B. Guler, A. S. Avestimehr, and P. Mohassel, "CodedPrivateML: A fast and privacy-preserving framework for distributed machine learning," 2019. [Online]. Available: arXiv:1902.00641.
[48] J. So, B. Guler, and A. S. Avestimehr, "A scalable approach for privacy-preserving collaborative machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020.
[49] T. M. Cover and J. A. Thomas, *Elements of Information Theory* (Telecommunications and Signal Processing). Hoboken, NJ, USA: Wiley-Intersci., 2006.
[50] C. He *et al.*, "FedML: A research library and benchmark for federated machine learning," 2020. [Online]. Available: arXiv:2007.13518.
[51] L. Dalcín, R. Paz, and M. Storti, "MPI for python," *J. Parallel Distrib. Comput.*, vol. 65, no. 9, pp. 1108–1115, 2005.
[52] J. So, B. Güler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE J. Sel. Areas Commun.*, early access, Dec. 2, 2020, doi: 10.1109/JSAC.2020.3041404.
[53] C. He, S. Avestimehr, and M. Annavaram, "Group knowledge transfer: Collaborative training of large CNNs on the edge," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020.
[54] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Stat. Assoc.*, vol. 58, no. 301, pp. 13–30, 1963.
[55] K. S. Kedlaya and C. Umans, "Fast polynomial factorization and modular composition," *SIAM J. Comput.*, vol. 40, no. 6, pp. 1767–1802, 2011.

Jinhyun So received the B.S. and M.S. degrees in electrical and computer engineering from KAIST. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Southern California. His research interests include information theory, large-scale distributed machine learning, and secure and private computing. He received the Annenberg Graduate Fellowship in 2017.

Başak Güler (Member, IEEE) received the Ph.D. degree from Pennsylvania State university in 2017. She was a Postdoctoral Scholar with the University of Southern California from 2018 to 2020. She is an Assistant Professor with the Department of Electrical and Computer Engineering, University of California at Riverside. Her research interests include machine learning in wireless networks, information theory, distributed computing, and signal processing.

A. Salman Avestimehr (Fellow, IEEE) received the B.S. degree in electrical engineering from the Sharif University of Technology in 2003, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California at Berkeley in 2005 and 2008, respectively.

He is a Professor and the Director of the Information Theory and Machine Learning (vITAL) Research Lab, Electrical and Computer Engineering Department, University of Southern California. His research interests include information theory and coding theory, and large-scale distributed computing and machine learning, secure and private computing, and blockchain systems. He has received a number of awards for his research, including the James L. Massey Research and Teaching Award from the IEEE Information Theory Society, the Information Theory Society and Communication Society Joint Paper Award, the Presidential Early Career Award for Scientists and Engineers (PECASE) from the White House (President Obama), the Young Investigator Program (YIP) Award from the U.S. Air Force Office of Scientific Research, the National Science Foundation CAREER Award, the David J. Sakrison Memorial Prize, and several best paper awards at conferences. He has been an Associate Editor of IEEE TRANSACTIONS ON INFORMATION THEORY and a General Co-Chair of the 2020 International Symposium on Information Theory.