Secure Aggregation for Buffered Asynchronous Federated Learning

Jinhyun So
ECE Department
University of Southern California (USC)
jinhyuns@usc.edu

Ramy E. Ali
ECE Department
University of Southern California (USC)
reali@usc.edu

Başak Güler
ECE Department
University of California, Riverside
bguler@ece.ucr.edu

A. Salman Avestimehr
ECE Department
University of Southern California (USC)
avestime@usc.edu

Abstract

Federated learning (FL) typically relies on synchronous training, which is slow due to stragglers. While asynchronous training handles stragglers efficiently, it does not ensure privacy due to the incompatibility with the secure aggregation protocols. A buffered asynchronous training protocol known as `FedBuff` has been proposed recently which bridges the gap between synchronous and asynchronous training to mitigate stragglers and to also ensure privacy simultaneously. `FedBuff` allows the users to send their updates asynchronously while ensuring privacy by storing the updates in a trusted execution environment (TEE) enabled private buffer. TEEs, however, have limited memory which limits the buffer size. Motivated by this limitation, we develop a buffered asynchronous secure aggregation (`BASecAgg`) protocol that does not rely on TEEs. The conventional secure aggregation protocols cannot be applied in the buffered asynchronous setting since the buffer may have local models corresponding to different rounds and hence the masks that the users use to protect their models may not cancel out. `BASecAgg` addresses this challenge by carefully designing the masks such that they cancel out even if they correspond to different rounds. Our convergence analysis and experiments show that `BASecAgg` almost has the same convergence guarantees as `FedBuff` without relying on TEEs.

1 Introduction

Federated learning (FL) allows users to collaboratively train a machine learning model without sharing their data and while protecting their privacy [18]. The training is typically coordinated by a central server. The main idea that enables decentralized training without sharing data is that each user trains a local model using its dataset and the global model maintained by the server. The users then only share their local models with the server which updates the global model and pushes it again to the users for the next training round until convergence. Recent studies, however, showed that sharing the local models still breaches the privacy of the users through inference or inversion attacks e.g., [10, 19, 30, 11]. To overcome this challenge, secure aggregation protocols were developed to ensure that the server only learns the global model without revealing the local models [4, 24, 13, 29, 9, 2]. FL protocols commonly rely on synchronous training [18], which suffers from stragglers due to waiting for the updates of a sufficient number of users at each round. Asynchronous FL tackles this by incorporating the updates of the users as soon as they arrive at the server [27, 26, 6, 7]. While asynchronous FL handles stragglers efficiently, it is not compatible with the secure aggregation protocols designed particularly for synchronous FL. This is because these protocols securely aggregate many local models together each time the global model is updated and hence they are not suitable for asynchronous FL in which each single local model updates the global model. Another approach that can be applied in asynchronous FL to protect the privacy of the users

1st NeurIPS Workshop on New Frontiers in Federated Learning (NFFL 2021), Virtual Meeting.

is local differential privacy (LDP) [25]. In this approach, each user adds a noise to the local model before sharing it with the server. This approach, however, degrades the training accuracy.

In [20], an asynchronous aggregation protocol known as `FedBuff` has been proposed to mitigate stragglers and enable secure aggregation jointly. `FedBuff` enables secure aggregation through trusted execution environments (TEEs) as Intel software guard extensions (SGX) [8]. Specifically, the individual updates are not incorporated by the server as soon they arrive. Instead, the server keeps the received local models in a TEE-enabled *secure buffer* of size K, where K is a tunable parameter. The server then updates the global model when the buffer is full. This idea has been shown to be 3.8 times faster than the conventional synchronous FL schemes.

Contributions. Since TEEs have limited memory, which limits the buffer size K, and are inefficient compared to the untrusted hardware [8], we instead develop a buffered asynchronous secure aggregation protocol that does not rely on TEEs. The main challenge of leveraging the conventional secure aggregation protocols in the buffered asynchronous setting is that the pairwise masks may not cancel out. This is because of the asynchronous nature of this setting which may result in local models of different rounds in the buffer, while the pairwise masks cancel out if they belong to the same round. This requires a careful design of the masks such that they can be cancelled even if they do not correspond to the same round. Specifically, our contributions are as follows.

1. We propose a buffered asynchronous secure aggregation protocol that extends a recently proposed synchronous secure aggregation protocol known as `LightSecAgg` [28] to this buffered asynchronous setting. The key idea of our protocol, `BASecAgg`, is that we design the masks such that they cancel out even if they correspond to different training rounds.
2. We extend the convergence analysis of [20] to the case where the local updates are quantized, which is necessary for the secure aggregation protocols to protect the privacy of the local updates.
3. Our extensive experiments on MNIST and CIFAR datasets show that `BASecAgg` almost has the same convergence guarantees as `FedBuff` despite the quantization.

Related Works. Secure aggregation protocols typically rely on exchanging pairwise random-seeds and secret sharing them to tolerate users' dropouts [4, 24, 13, 2]. The running time of such approaches, however, increases significantly with the number of dropped users since the server reconstructs the mask of each dropped user. Recently, a secure aggregation protocol known as `LightSecAgg` has been proposed to address this challenge [28]. In `LightSecAgg`, unlike the prior works, the server directly reconstructs the aggregate masks of all surviving users, which results in a much faster training. The protocol of [29] is based on the *one-shot* reconstruction idea, but it requires a trusted third party unlike `LightSecAgg`. Prior secure aggregation protocols [4, 24, 13, 2, 28, 29] are designed for synchronous FL [18], which suffer from stragglers. Asynchronous FL handles this problem by updating the global model as soon as the server receives any local model [27, 26, 6, 7]. The larger the staleness is of the local model, the greater is the error when updating the global model [27]. Various techniques have been proposed to mitigate this problem through staleness-aware averaging [27, 6, 21].

Asynchronous FL, however, is not compatible with secure aggregation. A potential approach to ensure privacy then is through DP approaches that add noise to the local models before sharing them with the server [26, 12]. Adding noise, however, degrades the training accuracy. In [20], an asynchronous aggregation protocol known as `FedBuff` has been proposed to mitigate stragglers while ensuring privacy. The key idea of `FedBuff` is that the server stores the local models in a TEE-enabled *secure buffer* of size K until the buffer is full and then securely aggregates them. Due to the memory limitations of TEEs, this approach is only feasible when K is small. This motivates us in this work to develop a buffered asynchronous secure aggregation protocol *without TEEs*.

2 Buffered Asynchronous Secure Aggregation

In this section, we provide a brief overview of `FedBuff` [20] and then illustrate the incompatibility of the conventional secure aggregation with asynchronous FL. Later on, we introduce `BASecAgg`. The goal in FL is to collaboratively learn a global model $\boldsymbol{x} \in \mathbb{R}^d$, using the local datasets of N users without sharing them. This problem is formulated as minimizing a global loss function as follows

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} F(\boldsymbol{x}) = \sum_{i=1}^{N} w_i F_i(\boldsymbol{x}), \tag{1}$$

where F_i is the local loss function of user $i \in [N]$ and $w_i \geq 0$ are the weight parameters that indicate the relative impact of the users and are selected such that $\sum_{i=1}^{N} w_i = 1$. This problem is solved iteratively in synchronous or asynchronous FL. We provide an overview of secure aggregation of synchronous FL in Appendix A and now discuss asynchronous FL.

In asynchronous FL, the updates of the users are not synchronized while the goal is the same as the synchronous FL to minimize the global loss function in (1). In the buffered asynchronous setting, the server stores each local model that it receives in a buffer of size K and updates the global model when the buffer is full. In our setting, this buffer is *not a secure buffer*. Hence, our goal is to design the secure aggregation protocol where users send the masked updates to protect the privacy in a way that the server can aggregate the local updates while the server (and potential colluding users) learns no information about the local updates beyond the aggregate of the updates stored in the buffer.

FedBuff. Before presenting our protocol, `BASecAgg`, we first provide an overview about the buffered asynchronous aggregation framework, named `FedBuff` [20], and describe the challenges that render `SecAgg` incompatible with this framework. The key intuition of `FedBuff` is to introduce a new design parameter K, the buffer size at the server, so that `FedBuff` has two degrees of freedom, K and the concurrency C while the synchronous FL frameworks have only one degree of freedom, concurrency. The concurrency is the number of users training concurrently and is an important parameter to provide a trade-off between the training time and the data inefficiency. Synchronous FL speeds up the training by increasing the concurrency, but higher concurrency results in data inefficiency [20]. In `FedBuff`, however, a high concurrency coupled with a proper value of K results in fast training. In other words, the additional degree of freedom K allows the server to update more frequently than concurrency, which enables `FedBuff` to achieve data efficiency at high concurrency.

At round t, C users are locally training the model by carrying out $E \geq 1$ local SGD steps. When the local update is done, user i sends the difference between the downloaded global model and updated local model to the server. The local update of user i sent to the server at round t is given by

$$\Delta_i^{(t;t_i)} = \boldsymbol{x}^{(t_i)} - \boldsymbol{x}_i^{(E)}, \tag{2}$$

where t_i is the latest round index when the global model is downloaded by user i and t is the round index when the local update is sent to the server, hence the staleness of user i is given by $\tau_i = t - t_i$. $\boldsymbol{x}_i^{(E)}$ denotes the local model after E local SGD steps and the local model at user i is updated as

$$\boldsymbol{x}_i^{(e)} = \boldsymbol{x}_i^{(e-1)} - \eta_l g_i(\boldsymbol{x}_i^{(e-1)}; \xi_i) \tag{3}$$

for $e = 1, \ldots, E$, where $\boldsymbol{x}_i^{(0)} = \boldsymbol{x}^{(t_i)}$, η_l denotes learning rate of the local updates. $g_i(\boldsymbol{x}; \xi_i)$ denotes the stochastic gradient with respect to the random sampling ξ_i on user i, and we assume $\mathbb{E}_{\xi_i}[g_i(\boldsymbol{x}; \xi_i)] = \nabla F_i(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^d$ where F_i is the local loss function of user i defined in (1). The server stores the received local updates in a buffer of size K. When the buffer is full, the server updates the global model by subtracting the aggregate of all local updates from the current global model. Specifically, the global model at the server is updated as

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \frac{\eta_g}{\sum_{i \in \mathcal{S}^{(t)}} s(t - t_i)} \sum_{i \in \mathcal{S}^{(t)}} s(t - t_i) \Delta_i^{(t;t_i)}, \tag{4}$$

where $\mathcal{S}^{(t)}$ is an index set of the K users whose local models are in the buffer at round t and η_g is the learning rate of the global updates. $s(\tau)$ is a function that compensates for the staleness satisfying $s(0) = 1$ and is monotonically decreasing as τ increases. There are many functions that satisfy these two properties and we consider a polynomial function $s_\alpha(\tau) = (\tau + 1)^{-\alpha}$ as it shows similar or better performance than the other functions e.g., Hinge or Constant stale function [27].

Privacy and Dropout Model. We assume at most D users may dropout in any round and a threat model where the users and the server are honest but curious who follow the protocol but try to infer the local updates of the other users. The secure aggregation protocol guarantees that nothing beyond the aggregate of the local updates is revealed, even if up to T users collude with the server. We consider information-theoretic privacy where from every subset of users $\mathcal{T} \subseteq [N]$ of size at most T, we must have mutual information $I(\{\Delta_i^{(t;t_i)}\}_{i \in [N]}; \mathbf{Y}^{(t)} | \sum_{i \in \mathcal{S}^{(t)}} \Delta_i^{(t;t_i)}, \mathbf{Z}_{\mathcal{T}}^{(t)}) = 0$, where $\mathbf{Y}^{(t)}$ and $\mathbf{Z}_{\mathcal{T}}^{(t)}$ are the collection of information at the server and at the users in $\mathcal{T}$ at round t, respectively.

2.1 Incompatibility of `SecAgg` with Buffered Asynchronous FL

As described in Appendix A.1, `SecAgg` [4] is designed for synchronous FL. At round t, each pair of users $i, j \in [N]$ agree on a pairwise random-seed $a_{i,j}^{(t)}$, and generate a random vector by running PRG based on the random seed of $a_{i,j}^{(t)}$ to mask the local update. This additive structure has the unique property that these pairwise random vectors cancel out when the server aggregates the masked models because user $i(< j)$ adds $\mathrm{PRG}(a_{i,j}^{(t)})$ to $\boldsymbol{x}_i^{(t)}$ and user $j(> i)$ subtracts $\mathrm{PRG}(a_{i,j}^{(t)})$ from $\boldsymbol{x}_j^{(t)}$.

In the buffered asynchronous FL, however, the cancellation of the pairwise random masks based on the key agreement protocol is not guaranteed due to the mismatch in staleness between users. Specifically, at round t, user $i \in \mathcal{S}^{(t)}$ sends the masked model $\boldsymbol{y}_i^{(t;t_i)}$ to the server that is given by

$$\boldsymbol{y}_i^{(t;t_i)} = \Delta_i^{(t;t_i)} + \mathrm{PRG}\left(b_i^{(t_i)}\right) + \sum_{j:i<j} \mathrm{PRG}\left(a_{i,j}^{(t_i)}\right) - \sum_{j:i>j} \mathrm{PRG}\left(a_{j,i}^{(t_i)}\right), \tag{5}$$

where $\Delta_i^{(t;t_i)}$ is the local update defined in (2). When $t_i \neq t_j$, the pairwise random vectors in $\boldsymbol{y}_i^{(t;t_i)}$ and $\boldsymbol{y}_j^{(t;t_j)}$ are not canceled out as $a_{i,j}^{(t_i)} \neq a_{i,j}^{(t_j)}$. We note that the identity of the staleness of each user is not known a priori, hence each pair of users cannot use the same pairwise random-seed.

2.2 The Proposed `BASecAgg` Protocol

To address the challenge of asynchrony in the buffered asynchronous secure aggregation, we propose `BASecAgg` by modifying the idea of *one-shot* recovery leveraged in `LightSecAgg` [28] to our setting. We provide a brief overview of `LightSecAgg` in Appendix A.2. Our key intuition is to encode the local masks in a way that the server can recover the aggregate of masks from the encoded masks via a one-shot computation even though the masks are generated in different training rounds.
Specifically, the users share the encoded masks with the time stamps to figure out which encoded masks should be aggregated for the reconstruction of the aggregate masks. Due to the commutative property of coding and addition, the server can reconstruct the aggregate masks even though the masks are generated in different training rounds. In particular, `BASecAgg` has three phases. First, each user generates a random mask to protect the privacy of the local update, and further creates encoded masks via a T*-private* Maximum Distance Separable (MDS) code that provides privacy against T colluding users. Each user sends one of the encoded masks to one of the other users for the purpose of one-shot recovery. Second, each user trains a local model and converts it from the domain of real numbers to the finite field as generating random masks and MDS encoding are required to be carried out in the finite field to provide information-theoretic privacy. Then, the quantized model is masked by the random mask generated in the first phase, and sent to the server. The server stores the masked update in the buffer. Third, when the buffer is full, the server aggregates the K masked updates in the buffer. To remove the randomness in the aggregate of the masked updates, the server reconstructs the aggregated masks of the users in the buffer. To do so, each surviving user sends the aggregate of encoded masks to the server. After receiving a sufficient number of aggregated encoded masks, the server reconstructs the aggregate of masks and hence the aggregate of the K local updates. We now describe these three phases in detail.

2.2.1 Offline Encoding and Sharing of Local Masks

User i generates $\boldsymbol{z}_i^{(t_i)}$ uniformly at random from the finite field $\mathbb{F}_q^d$, where t_i is the global round index when user i downloads the global model from the server. The mask $\boldsymbol{z}_i^{(t_i)}$ is partitioned into $U - T$ sub-masks denoted by $[\boldsymbol{z}_i^{(t_i)}]_1, \cdots, [\boldsymbol{z}_i^{(t_i)}]_{U-T}$, where U denotes the targeted number of surviving users and $N - D \geq U \geq T$. User i also selects another T random masks denoted by $[\boldsymbol{n}_i^{(t_i)}]_{U-T+1}, \cdots, [\boldsymbol{n}_i^{(t_i)}]_U$. These U partitions $[\boldsymbol{z}_i^{(t_i)}]_1, \cdots, [\boldsymbol{z}_i^{(t_i)}]_{U-T}, [\boldsymbol{n}_i^{(t_i)}]_{U-T+1}, \cdots, [\boldsymbol{n}_i^{(t_i)}]_U$ are then encoded through an (N, U) Maximum Distance Separable (MDS) code as follows

$$[\widetilde{\boldsymbol{z}}_i^{(t_i)}]_j = \left([\boldsymbol{z}_i^{(t_i)}]_1, \cdots, [\boldsymbol{z}_i^{(t_i)}]_{U-T}, [\boldsymbol{n}_i^{(t_i)}]_{U-T+1}, \cdots, [\boldsymbol{n}_i^{(t_i)}]_U\right) \boldsymbol{v}_j, \tag{6}$$

where $\boldsymbol{v}_j$ is the j-th column of a Vandermonde matrix $\mathbf{V} \in \mathbb{F}_q^{U \times N}$. After that, user i sends $[\widetilde{\boldsymbol{z}}_i^{(t_i)}]_j$ to user $j \in [N] \setminus \{i\}$. At the end of this phase, each user $i \in [N]$ has $[\widetilde{\boldsymbol{z}}_j^{(t_j)}]_i$ from $j \in [N]$.

2.2.2 Training, Quantizing, Masking, and Uploading of Local Updates

Each user i trains the local model as in (2) and (3). User i quantizes its local update $\Delta_i^{(t;t_i)}$ from the domain of real numbers to the finite field $\mathbb{F}_q$ as masking and MDS encoding are carried out in the finite field to provide information-theoretic privacy. The field size q is assumed to be large enough to avoid any wrap-around during secure aggregation.

The quantization is a challenging task as it should be performed in a way to ensure the convergence of the global model. Moreover, the quantization should allow the representation of negative integers in the finite field, and enable computations to be carried out in the quantized domain. Therefore, we cannot utilize well-known gradient quantization techniques such as in [1], which represents the sign of a negative number separately from its magnitude. `BASecAgg` addresses this challenge with a simple stochastic quantization strategy combined with the two's complement representation as described next. For any positive integer $c \geq 1$, we first define a stochastic rounding function as

$$Q_c(x) = \begin{cases} \frac{\lfloor cx \rfloor}{c} & \text{with prob. } 1 - (cx - \lfloor cx \rfloor) \\ \frac{\lfloor cx \rfloor + 1}{c} & \text{with prob. } cx - \lfloor cx \rfloor, \end{cases} \tag{7}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x, and this rounding function is unbiased, i.e., $\mathbb{E}_Q[Q_c(x)] = x$. The parameter c is a design parameter to determine the number of quantization levels. The variance of $Q_c(x)$ decreases as the value of c increases, which will be described in Lemma 1 in Appendix B in detail. We then define the quantized update

$$\overline{\Delta}_i^{(t;t_i)} := \phi\left(c_l \cdot Q_{c_l}\left(\Delta_i^{(t;t_i)}\right)\right), \tag{8}$$

where the function Q_c from (7) is carried out element-wise, and c_l is a positive integer parameter to determine the quantization level of the local updates. The mapping function $\phi : \mathbb{R} \to \mathbb{F}_q$ is defined to represent a negative integer in the finite field by using the two's complement representation,

$$\phi(x) = \begin{cases} x & \text{if } x \geq 0 \\ q + x & \text{if } x < 0. \end{cases} \tag{9}$$

To protect the privacy of the local updates, user i masks the quantized update $\overline{\Delta}_i^{(t;t_i)}$ in (8) as

$$\widetilde{\Delta}_i^{(t;t_i)} = \overline{\Delta}_i^{(t;t_i)} + \boldsymbol{z}_i^{(t_i)}, \tag{10}$$

and sends the pair of $\left\{\widetilde{\Delta}_i^{(t;t_i)}, t_i\right\}$ to the server. The local round index t_i will be used in two cases: (1) when the server identifies the staleness of each local update and compensates it, and (2) when the users aggregate the encoded masks for one-shot recovery, which will be explained in Section 2.2.3.

2.2.3 One-shot Aggregate-update Recovery and Global Model Update

The server stores $\widetilde{\Delta}_i^{(t;t_i)}$ in the buffer, and when the buffer of size K is full the server aggregates the K masked local updates. In this phase, the server intends to recover

$$\sum_{i \in \mathcal{S}^{(t)}} s(t - t_i) \Delta_i^{(t;t_i)}, \tag{11}$$

where $\Delta_i^{(t;t_i)}$ is the local update in the real domain defined in (2), $\mathcal{S}^{(t)}$ ($\left|\mathcal{S}^{(t)}\right| = K$) is the index set of users whose local updates are stored in the buffer and aggregated by the server at round t, and $s(\tau)$ is the staleness function defined in (4). To do so, the first step is to reconstruct $\sum_{i \in \mathcal{S}^{(t)}} s(t - t_i) \boldsymbol{z}_i^{(t_i)}$. This is challenging as the decoding should be performed in the finite field, but the value of $s(\tau)$ is a real number. To address this problem, we introduce a quantized staleness function $\overline{s} : \{0, 1, \ldots, \} \to \mathbb{F}_q$,

$$\overline{s}_{c_g}(\tau) = c_g Q_{c_g}\left(s(\tau)\right), \tag{12}$$

where $Q_c(\cdot)$ is a stochastic rounding function defined in (7), and c_g is a positive integer to determine the quantization level of staleness function. Then, the server broadcasts information of $\left\{\mathcal{S}^{(t)}, \{t_i\}_{i \in \mathcal{S}^{(t)}}, c_g\right\}$ to all surviving users. After identifying the selected users in $\mathcal{S}^{(t)}$, the local round indices $\{t_i\}_{i \in \mathcal{S}^{(t)}}$ and the corresponding staleness, user $j \in [N]$ aggregates its encoded sub-masks $\sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_g}(t - t_i) \left[\widetilde{\boldsymbol{z}}_i^{(t_i)}\right]_j$ and sends it to the server for the purpose of one-shot recovery. The key difference between `BASecAgg` and `LightSecAgg` is that in `BASecAgg`, the time stamp t_i for encoded masks $\left[\widetilde{\boldsymbol{z}}_i^{(t_i)}\right]_j$ for each $i \in \mathcal{S}^{(t)}$ can be different, hence user $j \in [N]$ must aggregate the encoded mask with the proper round index. Due to the commutative property of coding and linear operations, each $\sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_g}(t - t_i) \left[\widetilde{\boldsymbol{z}}_i^{(t_i)}\right]_j$ is an encoded version of $\sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_g}(t - t_i) \left[\boldsymbol{z}_i^{(t_i)}\right]_k$

for $k \in [U-T]$ using the MDS matrix (or Vandermonde matrix) $\mathbf{V}$ defined in (6). Thus, after receiving a set of any U results from surviving users in $\mathcal{U}_2$, where $|\mathcal{U}_2| = U$, the server reconstructs $\sum_{i\in\mathcal{S}^{(t)}} \overline{s}_{c_g}(t-t_i)\left[\boldsymbol{z}_i^{(t_i)}\right]_k$ for $k \in [U-T]$ via MDS decoding. By concatenating the $U-T$ aggregated sub-masks $\sum_{i\in\mathcal{S}^{(t)}} \overline{s}_{c_g}(t-t_i)\left[\boldsymbol{z}_i^{(t_i)}\right]_k$, the server can recover $\sum_{i\in\mathcal{S}^{(t)}} \overline{s}_{c_g}(t-t_i)\boldsymbol{z}_i^{(t_i)}$. Finally, the server obtains the desired global update as follows

$$\boldsymbol{g}^{(t)} = \frac{1}{c_g c_l \sum_{i\in\mathcal{S}^{(t)}} s_{c_g}(t-t_i)} \phi^{-1}\left(\sum_{i\in\mathcal{S}^{(t)}} \overline{s}_{c_g}(t-t_i)\widetilde{\overline{\Delta}}_i^{(t;t_i)} - \sum_{i\in\mathcal{S}^{(t)}} \overline{s}_{c_g}(t-t_i)\boldsymbol{z}_i^{(t_i)}\right), \quad (13)$$

where c_l is defined in (8) and $\phi^{-1}: \mathbb{F}_q \to \mathbb{R}$ is the demapping function defined as follows

$$\phi^{-1}(\overline{x}) = \begin{cases} \overline{x} & \text{if} \quad 0 \leq \overline{x} < \frac{q-1}{2} \\ \overline{x}-q & \text{if} \quad \frac{q-1}{2} \leq \overline{x} < q. \end{cases} \quad (14)$$

Finally, the server updates the global model as $\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta_g \boldsymbol{g}^{(t)}$, which is equivalent to

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \frac{\eta_g}{\sum_{i\in\mathcal{S}^{(t)}} Q_{c_g}(s(t-t_i))} \sum_{i\in\mathcal{S}^{(t)}} Q_{c_g}(s(t-t_i))\, Q_{c_l}\left(\Delta_i^{(t;t_i)}\right), \quad (15)$$

where Q_{c_l} and Q_{c_g} are the stochastic rounding function defined in (7) with respect to quantization parameters c_l and c_g, respectively.

3 Convergence Analysis

In this section, we provide the convergence guarantee of BASecAgg in the L-smooth and non-convex setting. For simplicity, we consider the constant staleness function $s(\tau) = 1$ for all τ in (15). Then, the global update equation of BASecAgg is given by

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \frac{\eta_g}{K} \sum_{i\in\mathcal{S}^{(t)}} Q_{c_l}\left(\Delta_i^{(t;t_i)}\right), \quad (16)$$

where Q_{c_l} is the stochastic round function defined in (7), c_l is the positive constant to determine the quantization level, and $\Delta_i^{(t;t_i)}$ is the local update of user i defined in (2). We first introduce our assumptions, which are commonly made in analyzing FL algorithms [16, 20, 22, 23].

Assumption 1. *(Unbiasedness of local SGD). For all $i \in [N]$ and $\boldsymbol{x} \in \mathbb{R}^d$, $\mathbb{E}_{\xi_i}[g_i(\boldsymbol{x};\xi_i)] = \nabla F_i(\boldsymbol{x})$ where $g_i(\boldsymbol{x};\xi_i)$ is the stochastic gradient estimator of user i defined in* (3).

Assumption 2. *(Lipschitz gradient). $F_1, \ldots, F_N$ in* (1) *are all L-smooth: for all $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$ and $i \in [N]$, $\|\nabla F_i(\boldsymbol{a}) - \nabla F_i(\boldsymbol{b})\|^2 \leq L\|\boldsymbol{a}-\boldsymbol{b}\|^2$.*

Assumption 3. *(Bounded variance of local and global gradients). The variance of the stochastic gradients at each user is bounded, i.e., $\mathbb{E}_{\xi_i}\|\nabla g_i(\boldsymbol{x};\xi_i) - \nabla F_i(\boldsymbol{x})\|^2 \leq \sigma_l^2$ for $i \in [N]$ and $\boldsymbol{x} \in \mathbb{R}^d$. For the global loss function $F(\boldsymbol{x})$ defined in* (1)*, $\frac{1}{N}\sum_{i=1}^N \|\nabla F_i(\boldsymbol{x}) - \nabla F(\boldsymbol{x})\|^2 \leq \sigma_g^2$ holds.*

Assumption 4. *(Bounded gradient). For all $i \in [N]$, $\|\nabla F_i(\boldsymbol{x})\|^2 \leq G$.*

In addition, we make an assumption on the staleness of the local updates under asynchrony [20].

Assumption 5. *(Bounded staleness). For each global round index t and all users $i \in [N]$, the delay $\tau_i^{(t)} = t - t_i$ is not larger than a certain threshold $\tau_{\max}$ where t_i is the latest round index when the global model is downloaded to user i.*

Now, we state our main result for the convergence guarantee of BASecAgg.

Theorem 1. *Selecting the constant learning rates η_l and η_g such that $\eta_l\eta_g KE \leq \frac{1}{L}$, the global model iterates in* (16) *achieve the following ergodic convergence rate*

$$\frac{1}{J}\sum_{t=0}^{J-1} \mathbb{E}\left[|\nabla F(\boldsymbol{x}^{(t)})|^2\right] \leq \frac{2F^*}{\eta_g\eta_l EKT} + \frac{L\eta_g\eta_l\sigma_{c_l}^2}{2} + 3L^2E^2\eta_l^2\left(\eta_g^2K^2\tau_{\max}^2\right)\sigma^2, \quad (17)$$

where $F^* = F(\boldsymbol{x}^{(0)}) - F(\boldsymbol{x}^*)$, $\sigma^2 = G + \sigma_g^2 + \sigma_{c_l}^2$, and $\sigma_{c_l}^2 = \frac{d}{4{c_l}^2} + \sigma_l^2$.

The proof of Theorem 1 is provided in Appendix B.

Remark 1. Theorem 1 shows that convergence rates of BASecAgg and FedBuff (see Corollary 1 in [20]) are the same except for the increased variance of the local updates due to the quantization noise in BASecAgg. The amount of the increased variance $\frac{d}{4{c_l}^2}$ in $\sigma_{c_l}^2 = \frac{d}{4{c_l}^2} + \sigma_l^2$ is negligible for large c_l, which will be demonstrated in our experiments in Section 4.

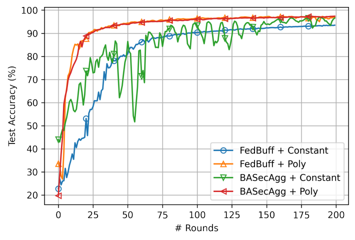
4 Experiments

In this section, we demonstrate the convergence performance of BASecAgg compared to the buffered asynchronous FL scheme from [20] termed FedBuff. We measure the performance in terms of the model accuracy evaluated over the test samples with respect to the global round index t.

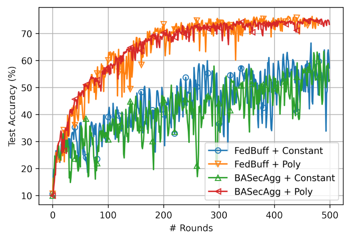
Datasets and network architectures. We consider an image classification task on the MNIST dataset [15] and CIFAR-10 dataset [14]. For MNIST dataset, we train LeNet [15]. For the CIFAR-10 dataset, we train the convolutional neural network (CNN) used in [27]. These network architectures are sufficient for our needs as our goal is to evaluate various schemes, not to achieve the best accuracy. More details about hyperparameters are provided in Appendix C. **Setup.** We consider a buffered asynchronous FL setting with $N = 100$ users and a single server having the buffer of size $K = 10$. For IID data distribution, the training samples are shuffled and partitioned into $N = 100$ users. For asynchronous training, we assume the staleness of each user is uniformly distributed over $[0, 10]$, i.e., $\tau_{\max} = 10$, as in [27]. We set the field size $q = 2^{32} - 5$, which is the largest prime within 32 bits.

Implementations. We implement two schemes, FedBuff and BASecAgg. The key difference between two schemes is that in BASecAgg, the local updates are quantized and converted into the finite field to provide privacy of the individual local updates while all operations are carried out over the domain of real numbers in FedBuff. For both schemes, to compensate the staleness of the local updates, we employ the two strategies for the weighting function: a constant function $s(\tau) = 1$ and a polynomial function $s_\alpha(\tau) = (1 + \tau)^{-\alpha}$.

Empirical results. In Figure 1(a) and 1(b), we demonstrate that BASecAgg has almost the same performance as FedBuff on both MNIST and CIFAR-10 datasets while BASecAgg includes quantization noise to protect the privacy of individual local updates of users. This is because the quantization noise in BASecAgg is negligible as explained in Remark 1. To compensate the staleness of the local updates over the finite field in BASecAgg, we implement the quantized staleness function defined in (12) with $c_g = 2^6$, which has the same performance to mitigate the staleness as the original staleness function carried out over the domain of real numbers.

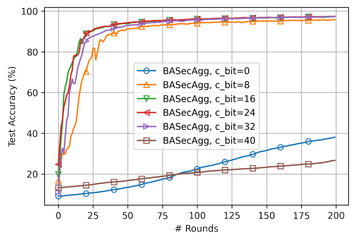


(a) MNIST dataset.

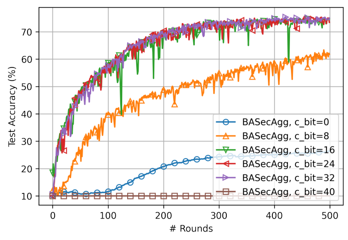


(b) CIFAR-10 dataset.

Figure 1: Accuracy of BASecAgg and FedBuff with two strategies for the weighting function to mitigate the staleness: a constant function $s(\tau) = 1$ (no compensation) named *Constant*; and a polynomial function $s_\alpha(\tau) = (1 + \tau)^{-\alpha}$ named *Poly* where $\alpha = 1$.



(a) MNIST dataset.



(b) CIFAR-10 dataset.

Figure 2: Accuracy of BASecAgg and FedBuff with various values of the quantization parameter $c_l = 2^{c_{bit}}$.

Performance with various quantization levels. To investigate the impact of the quantization, we measure the performance with various values of the quantization parameter c_l on MNIST and CIFAR-10 datasets in Fig. 2. We can observe that $c_l = 2^{16}$ has the best performance while small or large values of c_l have poor performance. This is because the value of c_l provides a trade-off between two sources of quantization noise: 1) the rounding error from the stochastic rounding function defined in (7) and 2) the wrap-around error when modulo operations are carried out in the finite field. When c_l has small value the rounding error is dominant while the wrap-around error is dominant when c_l has large value. To find a proper value of c_l, we can utilize the auto-tuning algorithm proposed in [5].

References

[1] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pages 1709–1720, 2017.

[2] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269, 2020.

[3] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

[4] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[5] Keith Bonawitz, Fariborz Salehi, Jakub Konečnỳ, Brendan McMahan, and Marco Gruteser. Federated learning with autotuned communication-efficient secure aggregation. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 1222–1226. IEEE, 2019.

[6] Zheng Chai, Yujing Chen, Liang Zhao, Yue Cheng, and Huzefa Rangwala. FedAt: A communication-efficient federated learning method with asynchronous tiers under non-iid data. *arXiv preprint arXiv:2010.05958*, 2020.

[7] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24. IEEE, 2020.

[8] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[9] Ahmed Roushdy Elkordy and A Salman Avestimehr. Secure aggregation with heterogeneous quantization in federated learning. *arXiv preprint arXiv:2009.14388*, 2020.

[10] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[11] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020.

[12] Bin Gu, An Xu, Zhouyuan Huo, Cheng Deng, and Heng Huang. Privacy-preserving asynchronous vertical federated learning algorithms for multiparty collaborative learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[13] Swanand Kadhe, Nived Rajaraman, O Ozan Koyluoglu, and Kannan Ramchandran. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2009.11248*, 2020.

[14] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[15] Yann LeCun. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[16] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2019.

[17] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.

[18] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Int. Conf. on Artificial Int. and Stat. (AISTATS)*, pages 1273–1282, 2017.

[19] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, pages 739–753. IEEE, 2019.

[20] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Michael Rabbat, Mani Malek Esmaeili, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. *arXiv preprint arXiv:2106.06639*, 2021.

[21] Jungwuk Park, Dong-Jun Han, Minseok Choi, and Jaekyun Moon. Sself: Robust federated learning against stragglers and adversaries. 2020.

[22] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[23] Jinhyun So, Ramy E Ali, Basak Guler, Jiantao Jiao, and Salman Avestimehr. Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning. *arXiv preprint arXiv:2106.03328*, 2021.

[24] Jinhyun So, Başak Güler, and A Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):479–489, 2021.

[25] Stacey Truex, Ling Liu, Ka-Ho Chow, Mehmet Emre Gursoy, and Wenqi Wei. Ldp-fed: Federated learning with local differential privacy. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, pages 61–66, 2020.

[26] Marten van Dijk, Nhuong V Nguyen, Toan N Nguyen, Lam M Nguyen, Quoc Tran-Dinh, and Phuong Ha Nguyen. Asynchronous federated learning with reduced number of rounds and with differential privacy from less aggregated gaussian noise. *arXiv preprint arXiv:2007.09208*, 2020.

[27] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[28] Chien-Sheng Yang, Jinhyun So, Chaoyang He, Songze Li, Qian Yu, and Salman Avestimehr. LightSecAgg: Rethinking secure aggregation in federated learning. *arXiv preprint arXiv:2109.14236*, 2021.

[29] Yizhou Zhao and Hua Sun. Information theoretic secure aggregation with user dropouts. *arXiv preprint arXiv:2101.07750*, 2021.

[30] Ligeng Zhu and Song Han. Deep leakage from gradients. In *Federated Learning*, pages 17–31. Springer, 2020.