Reconciling Security and Communication Efficiency in Federated Learning

Karthik Prasad*[†] Sayan Ghosh*[†] Graham Cormode[†]
Ilya Mironov[†] Ashkan Yousefpour[†] Pierre Stock[†]
[†]Meta AI

Abstract

Cross-device Federated Learning is an increasingly popular machine learning setting to train a model by leveraging a large population of client devices with high privacy and security guarantees. However, communication efficiency remains a major bottleneck when scaling federated learning to production environments, particularly due to bandwidth constraints during uplink communication. In this paper, we formalize and address the problem of compressing client-to-server model updates under the Secure Aggregation primitive, a core component of Federated Learning pipelines that allows the server to aggregate the client updates without accessing them individually. In particular, we adapt standard scalar quantization and pruning methods to Secure Aggregation and propose Secure Indexing, a variant of Secure Aggregation that supports quantization for extreme compression. We establish state-of-the-art results on LEAF benchmarks in a secure Federated Learning setup with up to 40$\times$ compression in uplink communication with no meaningful loss in utility compared to uncompressed baselines.

1 Introduction

Federated Learning (FL) is a distributed machine learning (ML) paradigm that trains a model across a number of participating entities holding local data samples. In this work, we focus on cross-device FL that harnesses a large number (up to hundreds of millions) of edge devices with disparate characteristics such as availability, compute, memory, or connectivity resources [31]. Two challenges to the success of cross-device FL are privacy and scalability. FL was originally motivated for improving privacy since data points remain on client devices. However, as with other forms of ML, information about training data can be extracted via membership inference or reconstruction attacks on a trained model [11, 12], or leaked through local updates [40, 19]. Consequently, Secure Aggregation (SECAGG) protocols were introduced to prevent the server from directly observing individual client updates, which is a major vector for information leakage [8, 27]. Additional mitigations such as Differential Privacy (DP) may be required to offer further protection against attacks [17, 1], as discussed in Section 6.

Ensuring scalability to populations of heterogeneous clients is the second challenge for FL. Indeed, wall-clock training times are highly correlated with increasing model and batch sizes [27], even with recent efforts such as FedBuff [41], and communication overhead between the server and clients dominates model convergence time.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Equal contribution. Correspondence to `pstock@fb.com`.

Consequently, compression techniques were used to reduce the communication bandwidth while maintaining model accuracy. However, a fundamental problem has been largely overlooked in the literature: in their native form, standard compression methods such as scalar quantization and pruning are not compatible with SECAGG. This makes it challenging to ensure both security and communication efficiency.

We address this gap by adapting compression techniques to make them compatible with SECAGG. We focus on compressing uplink updates from clients to the server for three reasons. First, uplink communication is more sensitive and so is subject to a high security bar, whereas downlink updates broadcast by the server are deemed public. Second, upload bandwidth is generally more restricted than download bandwidth. For instance, according to a recent FCC report, the ratio of download to upload speeds for DSL and cable providers[1] in the US ranges between $3\times$ to $20\times$ [18]. Efficient uplink communication brings several benefits beyond speeding up convergence: lowering communication cost reduces selection bias due to under-sampling clients with limited connectivity, improving fairness and inclusiveness. It shrinks the carbon footprint of FL, the fraction of which attributable to communication can reach 95% [45]. In summary, we present the following contributions:

- We highlight the fundamental mismatch between two critical components of the FL stack: SECAGG protocols and uplink compression mechanisms.
- We formulate solutions by imposing a linearity constraint on the decompression operator, as illustrated in Figure 1 in the case of TEE-based SECAGG.
- We adapt the popular scalar quantization and (random) pruning compression methods for compatibility with the FL stack that require no changes to the SECAGG protocol.
- For extreme uplink compression without compromising security, we propose Secure Indexing (SECIND), a variant of SECAGG that supports product quantization.

2 Related Work

Communication is identified as a primary efficiency bottleneck in FL, especially in the cross-device FL setting [31]. This has led to significant interest in reducing FL's communication requirements. In what follows, we refer to a local model update in distributed training as a gradient, including updates from multiple local training steps.

Efficient Distributed Optimization. There is a large body of literature on reducing the communication cost for distributed training. [49] proposes quantizing gradients to one bit while carrying the quantization error forward across mini-batches with error feedback. Similarly, [57] proposes layer-wise ternary gradients and [6] suggests using only the sign of the gradients. Gradient sparsity is another related area that is extensively studied [56, 2, 36, 46, 42]. For instance, [14] and [22] explore adapting the degree of sparsity to the distribution of local client data. Another method, QSGD, tunes the quantization level to trade possibly higher variance gradients for reduced communication bandwidth [3]. Researchers also studied structured and sketched model updates [32]. For example, [54] proposes expressing gradients as a linear combination of basis vectors common to all workers and [55] propose to cluster the gradients and to implement error correction on the client side. Besides gradient compression, other methods such as [52, 26] reduce the communication cost by partitioning the model such that each client learns a portion of it, while [24] proposes training small models and periodically distilling them to a larger central model. However, as detailed in Section 3 and below, most of the proposed methods are not readily compatible with SECAGG and cannot be used in secure FL.

Bi-directional Compression. In addition to uplink gradient compression, a line of work also focuses on downlink model compression. In a non-distributed setup, [61, 16] demonstrates that it is possible to meaningfully train with low bit-width models and gradients. In FL, [30] proposes adapting the model size to the device to reduce

[1]FL is typically restricted to using unmetered connections, usually over Wi-Fi [27].

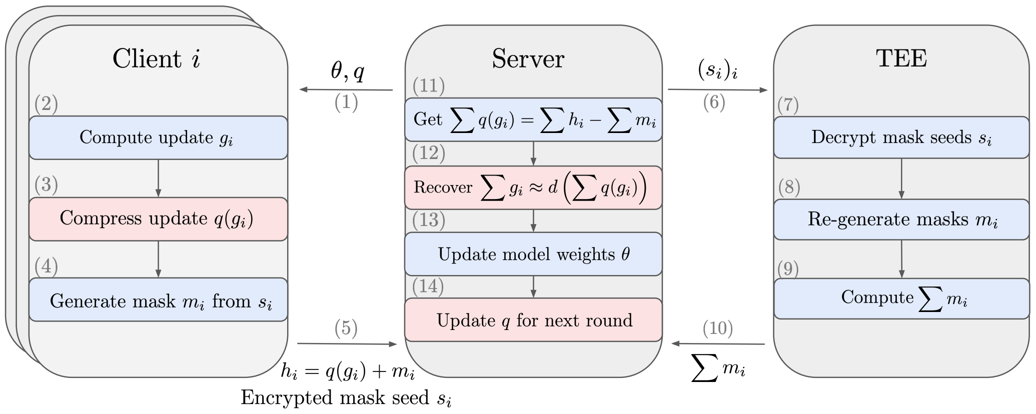

Figure 1: Summary of the proposed approach for one FL round, where we omit the round dependency and Differential Privacy (DP) for clarity. Blue boxes denote standard steps and red boxes denote additional steps for uplink compression. Client i computes local model update g_i, compresses it with the compression operator q, and encrypts it by adding a random mask m_i in the compressed domain, hence reducing the uplink bandwidth (steps 2–4). The server recovers the aggregate in the compressed domain by leveraging any SECAGG protocol (steps 7–13, with a TEE-based SECAGG, see Section 3.1). Since the decompression operator d is linear, the server can convert the aggregate back to the non-compressed domain, up to compression error (step 12). As with the model weights θ, the compression operator q are also periodically updated and broadcast by the server (step 14). In Section 4, we apply the proposed method to scalar quantization and pruning without impacting SECAGG and propose Secure Indexing, a variant of SECAGG for extreme uplink compression with product quantization. See Section 3.1 for details about SECAGG and Section 6 for a discussion on DP.

both communication and computation overhead. Since the local models are perturbed due to compression, researchers propose adapting the optimization algorithm for better convergence [37, 48, 51, 60, 4, 43]. Finally, pre-conditioning models during FL training can allow for quantized on-device inference, as demonstrated for non-distributed training by [21, 33]. As stated in Section 1, we do not focus on downlink model compression since uplink bandwidth is the main communication bottleneck and since SECAGG only involves uplink communication.
Aggregation in the Compressed Domain. In the distributed setting, [59] propose to leverage both gradient compression and parallel aggregation by performing the ring all-reduce operation in the compressed domain and decompressing the aggregate. To do so, the authors exploit temporal correlations of the gradients to design a linear compression operator. Another method, PowerSGD [53], leverages a fast low-rank gradient compressor. However, both aforementioned methods are not evaluated in the FL setup and do not mention SECAGG. Indeed, the proposed methods focus on decentralized communication between the workers by leveraging the all-reduce operation. Moreover, PowerSGD uses (stateful) error feedback on all distributed nodes, which is not readily adaptable to cross-device FL when clients generally participate in a few (not necessarily consecutive) rounds. Finally, [47] proposes FetchSGD, a compression method using sketching, which is compatible with SECAGG.

3 Background

In this section, we recall the SECAGG protocol first, then the compression methods that we wish to adapt to SECAGG, namely, scalar quantization, pruning, and product quantization.

3.1 Secure Aggregation

SECAGG refers to a class of protocols that allow the server to aggregate client updates without accessing them individually. While SECAGG alone does not entirely prevent client data leakage, it is a powerful and widely-used component of current at-scale cross-device FL implementations [31]. Two main approaches exist in practice: software-based protocols relying on Multiparty Computation (MPC) [8, 5, 58], and those that leverage hardware implementations of Trusted Execution Environments (TEEs) [27].

SecAgg relies on additive masking, where clients protect their model updates g_i by adding a uniform random mask m_i to it, guaranteeing that each client's masked update is statistically indistinguishable from any other value. At aggregation time, the protocol ensures that all the masks are canceled out. For instance, in an MPC-based SecAgg, the pairwise masks cancel out within the aggregation itself, since for every pair of users i and j, after they agree on a matched pair of input perturbations, the masks $m_{i,j}$ and $m_{j,i}$ are constructed so that $m_{i,j} = -m_{j,i}$. Similarly and as illustrated in Fig. 1, in a TEE-based SecAgg, the server receives $h_i = g_i + m_i$ from each client as well as the sum of the masks $\sum_i m_i$ from the TEE and recovers the sum of the updates as $\sum_i g_i = \sum_i h_i - \sum_i m_i$. We defer the discussion of DP noise addition by SecAgg protocols to Section 6.

Finite Group. SecAgg requires that the plaintexts—client model updates—be elements of a finite group, while the inputs are real-valued vectors represented with floating-point types. This requirement is usually addressed by converting client updates to fixed-point integers and operating in a finite domain (modulo 2^p) where p is typically set in prior literature to 32 bits. The choice of SecAgg bit-width p must balance communication costs with the accuracy loss due to rounding and overflows.

Minimal Complexity. TEE-based protocols offer greater flexibility in how individual client updates can be processed; however, the code executed inside TEE is part of the trusted computing base (TCB) for all clients. In particular, it means that this code must be stable, auditable, defects- and side-channel-free, which severely limits its complexity. Hence, in practice, we prefer compression techniques that are either oblivious to SecAgg's implementation or require minimal changes to the TCB.

3.2 Compression Methods

In this subsection, we consider a matrix $W \in \mathbb{R}^{C_{\text{in}} \times C_{\text{out}}}$ representing the weights of a linear layer to discuss three major compression methods with distinct compression/accuracy tradeoffs and identify the challenges SecAgg faces to be readily amenable to these popular quantization algorithms.

3.2.1 Scalar Quantization

Uniform scalar quantization maps floating-point weight w to 2^b evenly spaced bins, where b is the number of bits. Given a floating-point scale $s > 0$ and an integer shift parameter z called the zero-point, we map any floating-point parameter w to its nearest bin indexed by $\{0, \dots, 2^b - 1\}$:

$$w \mapsto \text{clamp}(\text{round}(w/s) + z, [0, 2^b - 1]).$$

The tuple (s, z) is often referred to as the quantization parameters (`qparams`). With $b = 8$, we recover the popular `int8` quantization scheme [28], while setting $b = 1$ yields the extreme case of binarization [16]. The quantization parameters s and z are usually calibrated after training a model with floating-point weights using the minimum and maximum values of each layer. The compressed representation of weights W consists of the `qparams` and the integer representation matrix W_q where each entry is stored in b bits. Decompressing any integer entry w_q of W_q back to floating point is performed by applying the (linear) operator $w_q \mapsto s \times (w_q - z)$.

Challenge. The discrete domain of quantized values and the finite group required by SecAgg are not natively compatible because of the overflows that may occur at aggregation time. For instance, consider the extreme case of binary quantization, where each value is replaced by a bit. We can represent these bits in SecAgg with $p = 1$, but the aggregation will inevitably result in overflows.

3.2.2 Pruning

Pruning is a class of methods that remove parts of a model such as connections or neurons according to some pruning criterion, such as weight magnitude ([34, 23]; see [7] for a survey). [32] demonstrate client update compression with random sparsity for federated learning. Motivated by previous work and the fact that random masks do not leak information about the data on client devices, we will leverage random pruning of client updates in the remainder of this paper. A standard method to store a sparse matrix is the coordinate list (COO) format[2], where only the non-zero entries are stored (in floating point or lower precision), along with their integer coordinates in the matrix. This format is compact, but only for a large enough compression ratio, as we store additional values for each non-zero entry. Decompression is performed by re-instantiating the uncompressed matrix with both sparse and non-sparse entries.

Challenge. Pruning model updates on the client side is an effective compression approach as investigated in previous work. However, the underlying assumption is that clients have different masks, either due to their seeds or dependency on client update parameters (*e.g.,* weight magnitudes). This is a challenge for SECAGG as aggregation assumes a dense compressed tensor, which is not possible to construct when the coordinates of non-zero entries are not the same for all clients.

3.2.3 Product Quantization

Product quantization (PQ) is a compression technique developed for nearest-neighbor search [29] that can be applied for model compression [50]. Here, we show how we can re-formulate PQ to represent model updates. We focus on linear layers and refer the reader to [50] for adaptation to convolutions. Let the block size be d (say, 8), the number of codewords be k (say, 256) and assume that the number of input channels, C_{in}, is a multiple of d. To compress W with PQ, we evenly split its columns into subvectors or blocks of size $d \times 1$ and learn a codebook via k-means to select the k codewords used to represent the $C_{\text{in}} \times C_{\text{out}}/d$ blocks of W. PQ with block size $d = 1$ amounts to non-uniform scalar quantization with $\log_2 k$ bits per weight.

The PQ-compressed matrix W is represented with the tuple (C, A), where C is the codebook of size $k \times d$ and A gives the assignments of size $C_{\text{in}} \times C_{\text{out}}/d$. Assignments are integers in $[0, k-1]$ and denote which codebook a subvector was assigned to. To decompress the matrix (up to reshaping), we index the codebook with the assignments, written in PyTorch-like notation as $\widehat{W} = C[A]$.

Challenge. There are several obstacles to making PQ compatible with SECAGG. First, each client may have a different codebook, and direct access to these codebooks is needed to decode each client's message. Even if all clients share a (public) codebook, the operation to take assignments to produce an (aggregated) update is not linear, and so cannot be directly wrapped inside SECAGG.

4 Method

In this section, we propose solutions to reconcile security (SECAGG) and communication efficiency. Our approach is to modify compression techniques to share some hyperparameters globally across all clients so that aggregation can be done by uniformly combining each client's response, while still ensuring that there is scope to achieve accurate compressed representations. As detailed below, each of the proposed methods offers the same level of security as standard SECAGG without compression.

4.1 Secure Aggregation and Compression

We propose to compress the uplink model updates through a compression operator q, whose parameters are round-dependent but the same for all clients participating in the same round. Then, we will add a random mask m_i to each quantized client update $q(g_i)$ in the compressed domain, thus effectively reducing uplink bandwidth while ensuring that $h_i = q(g_i) + m_i$ is statistically indistinguishable from any other representable value in the finite group (see Section 3.1). In this setting, SECAGG allows the server to recover the aggregate of the client

[2] See the torch.sparse documentation, `https://pytorch.org/docs/stable/sparse.html`.

model updates in the compressed domain: $\sum_i q(g_i)$. If the decompression operator d is linear, the server is able to recover the aggregate in the non-compressed domain, up to quantization error, as illustrated in Figure 1:

$$d\left(\sum_i h_i - \sum_i m_i\right) = d\left(\sum_i q(g_i)\right) = \sum_i d(q(g_i)) \approx \sum_i g_i.$$

The server periodically updates the quantization and decompression operator parameters, either from the aggregated model update, which is deemed public, or by emulating a client update on some similarly distributed public data. Once these parameters are updated, the server broadcasts them to the clients for the next round. This adds overhead to the downlink communication payload, however, this is negligible compared to the downlink model size to transmit. For instance, for scalar quantization, q is entirely characterized by one `fp32` scale and one `int32` zero-point per layer, the latter of which is unnecessary in the case of a symmetric quantization scheme. Finally, this approach is compatible with both synchronous FL methods such as FedAvg [39] and asynchronous methods such as FedBuff [41] as long as SECAGG maintains the mapping between the successive versions of quantization parameters and the corresponding client updates.

4.2 Application

Next, we show how we adapt scalar quantization and random pruning with no changes required to SECAGG. We illustrate our point with TEE-based SECAGG while these adapted uplink compression mechanisms are agnostic of the SECAGG mechanism. Finally, we show how to obtain extreme uplink compression by proposing a variant of SECAGG, which we call SECIND. This variant supports product quantization and is provably secure.

4.2.1 Scalar Quantization and Secure Aggregation

As detailed in Section 3.2.1, a model update matrix g_i compressed with scalar quantization is given by an integer representation in the range $[0, 2^b - 1]$ and by the quantization parameters $\underline{\text{scale}}$ (s) and $\underline{\text{zero-point}}$ (z). A sufficient condition for the decompression operator to be linear is to broadcast common quantization parameters per layer for each client. Denote $q(g_i)$ as the integer representation of quantized client model update g_i corresponding to a particular layer for client $1 \leq i \leq N$. Set the scale of the decompression operator to s and its zero-point to z/N. Then, the server is able to decompress as follows (where the decompression operator is defined in Section 3.2.1):

$$d\left(\sum_i q(g_i)\right) = s \sum_i q(g_i) - \frac{z}{N} = \sum_i \left(s(q(g_i)) - z\right) \approx \sum_i g_i$$

Recall that all operations are performed in a finite group. Therefore, to avoid overflows at aggregation time, we quantize with a bit-width b but take SECAGG bit-width $p > b$, thus creating a margin for potential overflows (see Section 5.3). This approach is related to the fixed-point aggregation described in [8, 27], but we calibrate the quantization parameters and perform the calibration per layer and periodically, unlike the related approaches.
Privacy, Security and Bandwidth. Scales and zero points are determined from public data on the server. Downlink overhead is negligible: the server broadcasts the per-layer quantization parameters. The upload bandwidth is p bits per weight, where p is the SECAGG finite group size (Section 3.1). Since the masks m_i are chosen in the integer range $[0, 2^p - 1]$, any masked integer representation taken modulo 2^p is statistically indistinguishable from any other vector.

4.2.2 Pruning and Secure Aggregation

To enable linear decompression with random pruning, all clients will share a common pruning mask for each round. This can be communicated compactly before each round as a seed for a pseudo-random function. This pruning mask seed is different from the SECAGG mask seed introduced in Section 3.1 and has a distinct role. Each client uses the pruning seed to reconstruct a pruning mask, prunes their model update g_i, and only needs to encrypt and transmit the unpruned parameters. The trade-off here is that some parameters are completely

Algorithm 1 Secure Indexing (SecInd)

```
1: procedure SecureIndexing(C)                              ▷ This happens inside the TEE
2:     Receive common codebook $C$ from server               ▷ $C$ is periodically updated by the server
3:     Initialize histograms $H_{m,n}$ to 0                  ▷ Each histogram for block $(m, n)$ has size $k$
4:     for each client $i$ do
5:         Receive and decrypt assignment matrix $A^i$
6:         for each block index $(m, n)$ do
7:             $r \leftarrow A^i_{m,n}$                      ▷ Recover assignment of client $i$ for block $(m, m)$
8:             $H_{m,n}[r] \leftarrow H_{m,n}[r] + 1$        ▷ Update global count for codeword index $r$
9:     Send back histograms $H_{m,n}$ to the server
```

unobserved in a given round, as opposed to traditional pruning. SecAgg operates as usual and the server receives the sum of the tensor of unpruned parameters computed by participating clients in the round, which it can expand using the mask seed. We denote the pruning operator as ϕ applied to the original model update g_i, and the decompression operator as d applied to a compressed tensor $\phi(g_i)$. Decompression is an expansion operation equivalent to multiplication with a sparse permutation matrix P_i whose entries are dependent on the i'th client's mask seed. Crucially, when all clients share the same mask seed within each round, we have $P_i = P$ for all i and linearity of decompression is maintained:

$$d\left(\sum_i \phi(g_i)\right) = P\left(\sum_i \phi(g_i)\right) = \sum_i P_i\phi(g_i) = \sum_i d(\phi(g_i)) \approx \sum_i g_i.$$

Privacy, Security and Bandwidth. Since the mask is random, no information leaks from the pruning mask. The downlink overhead (the server broadcasts one integer mask seed) is negligible. The upload bandwidth is simply the size of the sparse client model updates. Finally, there is no loss in security since each client uses standard SecAgg mechanism on the non-pruned entries.

4.2.3 Product Quantization and Secure Indexing

We next describe the Secure Indexing (SecInd) primitive, and discuss how to instantiate it. Recall that with PQ, each layer has its own codebook C as explained in Section 4. Let us fix one particular layer compressed with codebook C, containing k codewords. We assume that C is common to all clients participating in the round. Consider the assignment matrix of a given layer $(A^i)_{m,n}$ for client i. From these, we seek to build the <u>assignment histograms</u> $H_{m,n} \in \mathbb{R}^k$ that satisfy $H_{m,n}[r] = \sum_i \mathbf{1}\left(A^i_{m,n} = r\right)$, where the indicator function $\mathbf{1}$ satisfies $\mathbf{1}\left(A^i_{m,n} = r\right) = 1$ if $A^i_{m,n} = r$ and 0 otherwise. A <u>Secure Indexing</u> primitive will produce $H_{m,n}$ while ensuring that no other information about client assignments or partial aggregations is revealed. The server receives assignment histograms from SecInd and is able to recover the aggregated update for each block indexed by (m, n) as $\sum_r H_{m,n}[r] \cdot C[r]$. We describe how SecInd can be implemented with a TEE in Algorithm 1. Each client encrypts the assignment matrix, for instance with additive masking as described in Section 3.1, and sends it to the TEE via the server. Hence, the server does not have access to the plaintexts client-specific assignments. TEE decrypts each assignment matrix and for each block indexed by (m, n) produces the assignment histogram. Compared to SecAgg, where the TEE receives an encrypted seed per client (a few bytes per client) and sends back the sum of the masks m_i (same size as the considered model), SecInd receives the (masked) assignment matrices and sends back histograms for each round. SecInd can be implemented in other models, offering different trust paradigms, such as the multi-party computation setting (using two or more servers to operate on shares of the input). Encoding inputs as shares of one-hot vectors would lose the advantages of compression. Instead, each client can send evaluations of <u>distributed point functions</u> to encode each assignment [9]. These are represented compactly, but may require longer codewords to overcome the overheads.

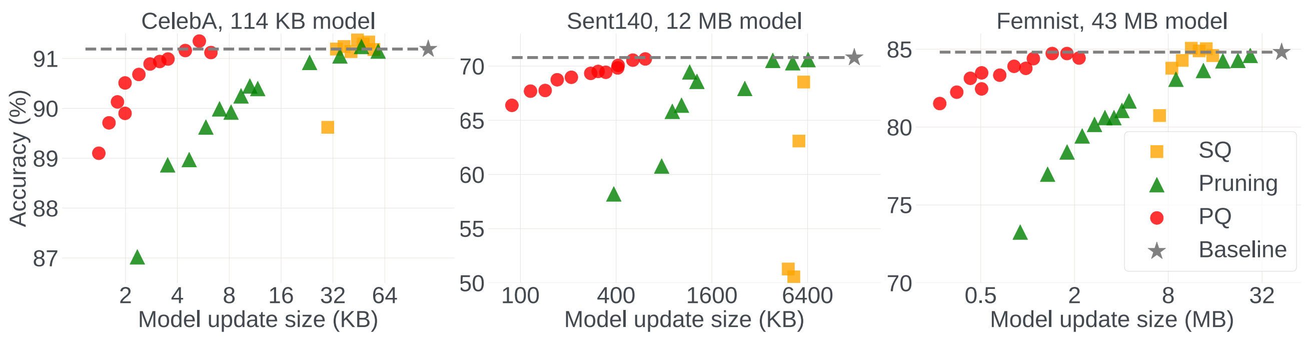

Figure 2: We adapt scalar quantization (SQ) and pruning to the SECAGG protocol to enable efficient and secure uplink communications. We also present results for product quantization (PQ) under the proposed novel SECIND protocol. The x axis is log-scale and represents the uplink message size. Baseline refers to SECAGG FL run without any uplink compression, shown as a horizontal line for easier comparison. Model size is indicated in plot titles. Uncompressed client updates are as large as the models when $p = 32$ (see Sec 3.1, shown as stars).

Privacy, Security and Bandwidth. Codebooks are computed from public data while individual assignments are never revealed to the server. The downlink overhead of sending the codebooks is negligible as demonstrated in Section 5. The upload bandwidth in the TEE implementation is the assignment size, represented in k bits (the number of codewords). For instance, with a block size $d = 8$ and $k = 32$ codewords, assignment storage costs are 5 bits per 8 weights, which converts to 0.625 bits per weight. The tradeoff compared to non-secure PQ is the restriction to a global codebook for all clients (instead of one tailored to each client), and the need to instantiate SECIND instead of SECAGG. Since the assignments are encrypted before being sent to the TEE, there is no loss in security. Here, any encryption mechanism (not necessarily relying on additive masking) would work.

5 Experiments

We evaluate the performance of the proposed approaches when adapted to SECAGG protocols. We study the relationship between uplink compression and model accuracy for the LEAF benchmark tasks. For scalar and product quantization we also analyze the impact of refresh rate for compression parameters on model performance.

5.1 Experimental Setup

We follow the setup of [41] and use the FLSim library for our experiments . All experiments are run on a single V100 GPU 16 GB (except for Sent140 where we use one V100 32 GB) and typically take a few hours to run.
Tasks. We run experiments on three datasets from the LEAF benchmark [10]: CelebA [38], Sent140 [20] and FEMNIST [35]. For CelebA, we train the same convolutional classifier as [41] with BatchNorm layers replaced by GroupNorm layers and 9,343 clients. For Sent140, we train an LSTM classifier for binary sentiment analysis with $59,400$ clients. For FEMNIST, we train a GroupNorm version of the ResNet18 [25] for digit classification with 3,550 clients. We do not compress biases and norm layers due to their small overhead.
Baselines. We focus here on the (synchronous) FedAvg approach although, as explained in Section 4, the proposed compression methods can be readily adapted to asynchronous FL aggregation protocols. As in prior work, we keep the number of clients per round to at most 100, a small fraction of the total considered population size [15, 13]. We report the average and standard deviation of accuracy over three independent runs for all tasks at different uplink byte sizes corresponding to various configurations of the compression operator.
Implementation Details. We refer the reader to [44] for full experiment details. The downlink overhead of sending the per-layer codebooks for product quantization is negligible. The convergence time in terms of rounds

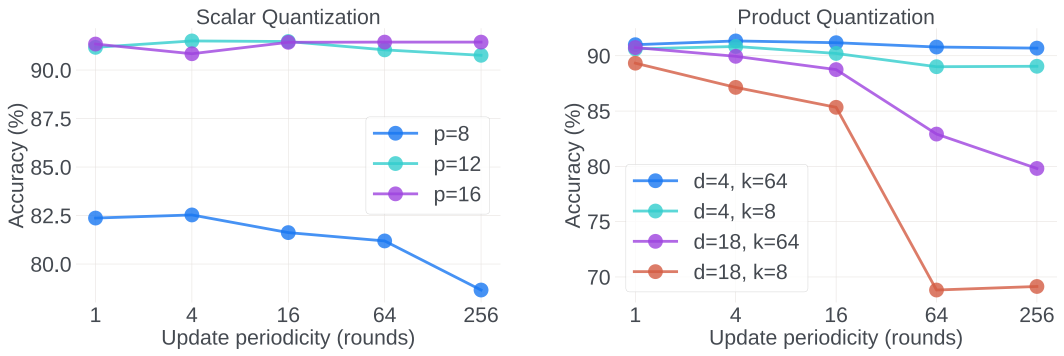

Figure 3: Impact of the refresh rate of the compression operator by the server on the CelebA dataset. **Left**: scalar quantization (quantization parameters), fixing the quantization bit-width $b = 8$ (p denotes the SECAGG bit-width). **Right**: for product quantization (codebooks), where k denotes the number of codewords and d the block size.

is similar for PQ runs and the non-compressed baseline/ Note that outside a simulated environment, the wall-clock time convergence for PQ runs would be lower than the baseline since uplink communication would be more efficient, hence faster.

5.2 Results and Comparison with Prior Work

Results for efficient and secure uplink communications are displayed in Figure 2. We observe that PQ yields a consistently better trade-off curve between model update size and accuracy. For instance, on CelebA, PQ achieves $\times 30$ compression with respect to the non-compressed baseline at iso-accuracy. The iso-accuracy compression rate is $\times 32$ on Sent140 and $\times 40$ on FEMNIST (see [44] for detailed tables). Scalar quantization accuracy degrades significantly for larger compression rates due to the overflows at aggregation. Pruning gives intermediate tradeoffs between scalar quantization and product quantization.

The line of work that develops FL compression techniques is exemplified by FetchSGD [47] as detailed in Section 2, where the authors do not address SECAGG. Their results are not directly comparable to ours due to incomparable experimental setups (e.g., datasets and architectures). However, Figure 6 [47, Appendix A] mentions upload compression rates at iso-accuracy that are weaker than those obtained here with product quantization.

5.3 Ablation Studies

We investigate the influence of the frequency of updates of the compression operator q for scalar quantization and pruning, and study the influence of the SECAGG bit-width p on the number of overflows for scalar quantization.

Update frequency of the compression operators. In Figure 3, we show that for scalar quantization, the update periodicity only plays a role with low SECAGG bit-width values p compared to the quantization bit-width b. For product quantization, the update periodicity plays an important role for aggressive compression setups corresponding to large block sizes d or to a smaller number of codewords k. For pruning, we measure the impact of masks that are refreshed periodically. We observed that if we refresh the compression operator more frequently, staleness is reduced, leading to accuracy improvements.

Overflows for scalar quantization. As discussed in Section 4.2.1, we choose the SECAGG bit-width p to be greater than quantization bit-width b in order to avoid aggregation overflows. While it suffices to set p to be $\lceil \log_2 n_c \rceil$ more than b, where n_c is the number of clients participating in the round, reducing p is desirable to reduce uplink size. We studied the impact of p on the percentage of parameters that suffer overflows, and observed that there is a benefit to having some non-zero overflow margin size, but no clear correlation between margin size and accuracy.

6 Concluding Remarks

In this paper, we reconcile efficiency and security for uplink communication in Federated Learning. We propose to adapt existing compression mechanisms such as scalar quantization and pruning to the secure aggregation protocol by imposing a linearity constraint on the decompression operator. Our experiments demonstrate that we can adapt both quantization and pruning mechanisms to obtain a high degree of uplink compression with minimal degradation in performance and higher security guarantees. For achieving the highest rates of compression, we introduce SECIND, a variant of SECAGG well-suited for TEE-based implementation that supports product quantization while maintaining a high security bar. As mentioned in Section 1, we may want both SECAGG and Differential Privacy [1] to realize the full promise of FL as a privacy-enhancing technology. While our primary focus is on enabling efficient and secure uplink communication, we emphasize that the proposed approaches are compatible with user-level DP. For instance, at the cost of increasing the complexity of the trusted computing base, DP noise can be added natively by the TEE with our modified random pruning or scalar quantization approaches. For PQ and SECIND, we can have the TEE to add noise in the assignment space (*i.e.,* outputting a noisy histogram), or to map the histogram to the codeword space and add noise there. Each option offers a different tradeoff between privacy, trust, and accuracy; we leave detailed evaluation to future work.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In ACM CCS, 2016.

[2] A. F. Aji and K. Heafield. Sparse communication for distributed gradient descent. In EMNLP, 2017.

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

[4] M. M. Amiri, D. Gunduz, S. R. Kulkarni, and H. V. Poor. Federated learning with quantized global model updates. CoRR, 2006.10672, 2020.

[5] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In ACM CCS, 2020.

[6] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar. signSGD: Compressed optimisation for non-convex problems. In ICML, PMLR, 2018.

[7] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? In MLSys, 2020.

[8] K. A. Bonawitz, F. Salehi, J. Konečný, B. McMahan, and M. Gruteser. Federated learning with autotuned communication-efficient secure aggregation. In ACSCC. IEEE, 2019.

[9] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. In ACM CCS, 2016.

[10] S. Caldas, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar. LEAF: A benchmark for federated settings. CoRR, abs/1812.01097, 2018.

[11] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramèr. Membership inference attacks from first principles. CoRR, abs/2112.03570, 2021.

[12] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. B. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel. Extracting training data from large language models. In USENIX Security Symposium, 2021.

[13] Z. Charles, Z. Garrett, Z. Huo, S. Shmulyian, and V. Smith. On large-cohort training for federated learning. CoRR, 2106.07820, 2021.

[14] C. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan. AdaComp: Adaptive residual gradient compression for data-parallel distributed training. In AAAI, 2018.

[15] M. Chen, R. Mathews, T. Ouyang, and F. Beaufays. Federated learning of out-of-vocabulary words. CoRR, 1903.10635, 2019.

[16] M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. CoRR, 1511.00363, 2015.

[17] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In Proceedings of the Third Conference on Theory of Cryptography, 2006.

[18] FCC. The eleventh Measuring Broadband America fixed broadband report, 2021.

[19] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller. Inverting gradients—How easy is it to break privacy in federated learning? In NeurIPS, 2020.

[20] A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 2009.

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In ICML, 2015.

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

[23] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal Brain Surgeon. In NeurIPS, 1992.

[24] C. He, M. Annavaram, and S. Avestimehr. Group knowledge transfer: Federated learning of large CNNs at the edge. In NeurIPS, 2020.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In IEEE CVPR, 2016.

[26] C. Hu, W. Bao, D. Wang, and F. Liu. Dynamic adaptive DNN surgery for inference acceleration on the edge. IEEE INFOCOM, 2019.

[27] D. Huba, J. Nguyen, K. Malik, R. Zhu, M. Rabbat, A. Yousefpour, C.-J. Wu, H. Zhan, P. Ustinov, H. Srinivas, K. Wang, A. Shoumikhin, J. Min, and M. Malek. Papaya: Practical, private, and scalable federated learning. In MLSys, 2022.

[28] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In IEEE CVPR, June 2018.

[29] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell., 33(1):117–128, 2011.

[30] Y. Jiang, S. Wang, B. J. Ko, W. Lee, and L. Tassiulas. Model pruning enables efficient federated learning on edge devices. CoRR, abs/1909.12326, 2019.

[31] P. Kairouz et al. Advances and open problems in federated learning. Found. Trends Mach. Learn., 14(1–2), 2021.

[32] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. In NIPS Workshop on Private Multi-Party Machine Learning, 2016.

[33] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference. CoRR, 1806.08342, 2018.

[34] Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In NeurIPS, 1989.

[35] Y. LeCun and C. Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[36] Y. Lin, S. Han, H. Mao, Y. Wang, and W. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In ICLR, 2018.

[37] X. Liu, Y. Li, J. Tang, and M. Yan. A double residual compression algorithm for efficient distributed learning. In AISTATS, 2020.

[38] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In IEEE ICCV, 2015.

[39] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In AISTATS, 2017.

[40] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. In IEEE S&P, 2019.

[41] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba. Federated learning with buffered asynchronous aggregation. In AISTATS, 2022.

[42] T. Parcollet, J. Fernandez-Marques, P. PB Gusmao, Y. Gao, and N. D. Lane. ZeroFL: Efficient on-device training for federated learning with local sparsity. In ICLR, 2022.

[43] C. Philippenko and A. Dieuleveut. Preserved central model for faster bidirectional compression in distributed settings. In NeurIPS, 2021.

[44] K. Prasad, S. Ghosh, G. Cormode, I. Mironov, A. Yousefpour, and P. Stock. Reconciling security and communication efficiency in federated learning. CoRR, 2207.12779, 2022.

[45] X. Qiu, T. Parcollet, J. Fernandez-Marques, P. P. B. de Gusmao, D. J. Beutel, T. Topal, A. Mathur, and N. D. Lane. A first look into the carbon footprint of federated learning. arXiv, 2102.07627, 2021.

[46] C. Renggli, S. Ashkboos, M. Aghagolzadeh, D. Alistarh, and T. Hoefler. SparCML: High-performance sparse communication for machine learning. In SC, 2019.

[47] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora. FetchSGD: Communication-efficient federated learning with sketching. In ICML, 2020.

[48] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek. Robust and communication-efficient federated learning from non-i.i.d. data. IEEE Transactions on Neural Networks and Learning Systems, 31:3400–3413, 2020.

[49] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In INTERSPEECH, 2014.

[50] P. Stock, A. Joulin, R. Gribonval, B. Graham, and H. Jégou. And the bit goes down: Revisiting the quantization of neural networks. In ICLR, 2020.

[51] H. Tang, C. Yu, X. Lian, T. Zhang, and J. Liu. DOUBLESQUEEZE: Parallel stochastic gradient descent with double-pass error-compensated compression. In ICML, 2019.

[52] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar. Split learning for health: Distributed deep learning without sharing raw patient data, 2018.

[53] T. Vogels, S. P. Karimireddy, and M. Jaggi. PowerSGD: Practical low-rank gradient compression for distributed optimization. In NeurIPS, 2019.

[54] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright. ATOMO: Communication-efficient learning via atomic sparsification. In NeurIPS, 2018.

[55] J. Wang, H. Qi, A. S. Rawat, S. Reddi, S. Waghmare, F. X. Yu, and G. Joshi. Fedlite: A scalable approach for federated learning on resource-constrained clients. CoRR, 2201.11865, 2022.

[56] J. Wangni, J. Wang, J. Liu, and T. Zhang. Gradient sparsification for communication-efficient distributed optimization. In NeurIPS, 2018.

[57] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. TernGrad: Ternary gradients to reduce communication in distributed deep learning. In NeurIPS, 2017.

[58] C. Yang, J. So, C. He, S. Li, Q. Yu, and S. Avestimehr. LightSecAgg: Rethinking secure aggregation in federated learning. In MLSys, 2022.

[59] M. Yu, Z. Lin, K. Narra, S. Li, Y. Li, N. S. Kim, A. Schwing, M. Annavaram, and S. Avestimehr. GradiVeQ: Vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training. In NeurIPS, 2018.

[60] S. Zheng, Z. Huang, and J. T. Kwok. Communication-efficient distributed blockwise momentum SGD with error-feedback. In NeurIPS, 2019.

[61] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR, abs/1606.06160, 2016.