ScionFL: Secure Quantized Aggregation for Federated Learning

Yaniv Ben-Itzhak†, Helen Möllering‡, Benny Pinkas*, Thomas Schneider‡, Ajith Suresh*, Oleksandr Tkachenko*, Shay Vargaftik†, Christian Weinert‡, Hossein Yalame‡, and Avishay Yanai†

†VMWare Research, ‡Technical University of Darmstadt, *Bar-Ilan University, †DFINITY Foundation, ‡Royal Holloway, University of London

Abstract

Privacy concerns in federated learning (FL) are commonly addressed with secure aggregation schemes that prevent a central party from observing plaintext client updates. However, most such schemes neglect orthogonal FL research that aims at reducing communication between clients and the aggregator and is instrumental in facilitating cross-device FL with thousands and even millions of (mobile) participants. In particular, quantization techniques can typically reduce client-server communication by a factor of $32 \times$.

In this paper, we unite both research directions by introducing an efficient secure aggregation framework based on outsourced multi-party computation (MPC) that supports any linear quantization scheme. Specifically, we design a novel approximate version of an MPC-based secure aggregation protocol with support for multiple stochastic quantization schemes, including ones that utilize the randomized Hadamard transform and Kashin’s representation. In our empirical performance evaluation, we show that with no additional overhead for clients and moderate inter-server communication, we achieve similar training accuracy as insecure schemes for standard FL benchmarks.

Beyond this, we present an efficient extension to our secure quantized aggregation framework that effectively defends against state-of-the-art untargeted poisoning attacks.

1 Introduction

Federated learning (FL) [89], where in each training round a subset of clients locally updates a global model that is then centrally aggregated, quickly became a leading paradigm for large-scale distributed machine learning training with decentralized data due to its promises of data privacy, resource efficiency, and ability to handle dynamic participants.

However, in terms of privacy, the central aggregator learns the individual client updates in the clear and thus can infer sensitive details about the clients’ private input data [53, 90, 104, 114, 125]. Hence, many secure aggregation schemes, e.g., [23, 51, 94], have been developed, where the central aggregator only learns the aggregation result, i.e., the global model (we refer the reader to [65, 98] for a discussion of differential privacy as another paradigm to protect privacy in FL). Most prominently, in the “SegAgg” protocol [23], clients exchange masks with peers to blind their model updates such that the masks cancel out during aggregation and reveal only the exact result. However, this approach requires an interactive setup between clients and thus is less reliable when dealing with real-world problems such as client dropouts (except for special variants [64, 81, 128]).

In terms of resource efficiency of plain FL, clients have to send parameter updates (also known as gradients), which typically consist of thousands or millions of coordinates with one floating point number per coordinate. For cross-device FL involving mobile clients with limited upload bandwidth, this quickly becomes infeasible when dealing with increasingly large model architectures, where gradients consist of millions of coordinates. Therefore, quantization schemes that exploit the noise resiliency of gradient-based FL methods (e.g., federated averaging [89]) have been proposed to significantly compress client updates (typically replacing the representation of each coordinate by a single bit, instead of a 32-bit floating point number), e.g., [4, 11, 17, 28, 111, 120, 123].

Unfortunately, so far the ML and security research communities have mostly worked individually on optimizing FL in terms of secure aggregation for privacy and quantization for resource efficiency. One of the very few exceptions is a work [33] that combines the SegAgg approach with sketching [108] for compression of gradients. Besides the mentioned disadvantage of SegAgg, their work considers only a single compression method. However, due to the trade-offs between accuracy and computational efficiency provided by different forms of quantization, which become especially relevant for gradients with millions of coordinates, it is important to have a modular approach.

In our work, we instead enable secure aggregation on quantized client updates considering a distributed aggregator to avoid a single point of trust. In this model, clients apply information-theoretically secure and computationally efficient secret-sharing techniques to distribute their sensitive updates.
among multiple servers such that none of the servers learns any information except for the final aggregation result. This model is well established in other recent works on secure aggregation (that do not consider quantized inputs) due to its manifold performance benefits [94, 97] and as it does not require an interactive setup between clients. We additionally emphasize that it corresponds well to trending multi-cloud and sky computing strategies [117], where organizations simultaneously utilize multiple infrastructure providers, which potentially also operate in different jurisdictions, thereby reducing the risk of having to disclose sensitive user data even upon receiving subpoena requests. Furthermore, this distributed computing model is useful when multiple organizations want to cooperate to jointly train with clients.

1.1 Our Contributions

In this paper, we make several contributions to enable efficient secure aggregation for FL with a distributed aggregator on quantized client updates.

1. Minimal Client Communication. Most importantly, our secure aggregation framework, which we call ScionFL, incurs no additional overhead in terms of client communication (modulo a small constant number of bytes) compared to insecure quantized schemes. We support a range of state-of-the-art quantization schemes, thereby allowing for use-case specific optimizations in terms of accuracy and run-time.

For this, we rely on outsourced multi-party computation (MPC), where clients secret-share their sensitive (quantized) model updates among servers that use interactive cryptographic protocols to securely compute the aggregation and then return the updated global model.

2. Unbiased Approximated Aggregation for Minimizing Inter-server Communication. Since network traffic is costly even among powerful servers [59], we further optimize inter-server communication by studying approximate MPC variants that exploit the noise tolerance of FL and might be of independent interest. Specifically, we extend a SPDZ-style dishonest majority MPC protocol [42] with an approximate conversion of Boolean-shared bit values to arithmetically-shared fixed point values. We formally prove that our resulting secure aggregation scheme is an unbiased estimator of the arithmetic mean and explore trade-offs between efficiency and accuracy that arise. Besides secure aggregation for FL, we point out that our approximate conversion approach can also result in significant performance improvements for systems such as PRIO [36], which is deployed by the ISRG [1], and PRIO+ [2], that currently rely on expensive exact conversions for privacy-preserving aggregation of user statistics.

3. Benchmarking Framework. We implement a combined end-to-end FL evaluation and MPC simulation environment. It allows to assess the performance and accuracy of our solution for various stochastic quantization schemes, including recent state-of-the-art distributed mean estimation techniques utilizing the randomized Hadamard transform [120] and Kashin’s representation [28].

Our results demonstrate that standard FL benchmarks’ accuracy is barely impacted while our approximation and optimizations can significantly reduce inter-server communication. For example, when training the LeNet architecture for image classification on the MNIST data set [80] using 1-bit stochastic quantization with Kashin’s representation [28], training accuracy is only slightly reduced from 99.04% to 98.71% after 1000 rounds, while inter-server communication drops from 16.44GB to 0.94GB compared to naive MPC-based secure aggregation when considering 500 clients per round, an improvement by factor 17.51 ×.

4. Untargeted Poisoning Defense. Since a minority of clients may act maliciously and try to degrade accuracy with their updates, we additionally design an efficient defense mechanism, which builds upon ScionFL to ensure the robustness of our framework. Our novel bipartite defense ScionFL-Aura defends against state-of-the-art untargeted poisoning attacks [112], combining magnitude clipping and directional filtering based on the gradients’ approximate L2-norms and cosine distances. Our optimizations and approximations enable a highly efficient realization in MPC.

In our empirical evaluation, we also implement and reproduce the results of the state-of-the-art Min-Max untargeted poisoning attack [112]. We can show that ScionFL-Aura effectively diminishes the impact of the attack with 20% malicious clients: it consistently filters out most of the manipulated updates and reduces the attack impact to less than 5% for almost all training iterations when quantization is in place.

1.2 Related Work

We present a summary of the most relevant related works here, with a more comprehensive discussion in App. A.

Compression Techniques. To reduce communication costs, there are three main directions for gradient compression in cross-device FL: (1) gradient sparsification (e.g., [3, 50, 73, 116]), (2) client-side memory and error-feedback (e.g., [4, 18, 106, 111]), and (3) entropy encodings (e.g., [4, 120, 122]). Reviews of current state-of-the-art gradient compression techniques and some open challenges can be found in [65, 73, 124].

Secure Aggregation & Model Inference Attacks. Several works [22, 79, 96] have shown the susceptibility of FL to inference attacks extracting confidential information about the used private training data. Secure aggregation [15, 23, 33, 36, 51] effectively defends against those attacks by prohibiting access to individual updates. However, existing schemes incur significant computation and communication overhead [51]. To the best of our knowledge, only Chen et al. [33] and Beguier et al. [14] combine gradient compression with secure aggregation. While [33] combines sketching [108] with differential privacy and the SegAgg protocol [23], the work of [14] combines quantization [17] with additive secret-sharing, but with
non-optimal client-server communication overhead. More detailed comparisons among secure aggregation schemes for FL can be found in [51, 87].

Poisoning Attacks & Defenses. Next to privacy issues, FL has also been shown vulnerable to manipulations by malicious clients [9, 10, 49, 112, 131]. Targeted attacks (also called backdoor attacks) aim at manipulating the inference results for specific attacker-chosen input samples [9, 131]. In contrast, untargeted poisoning attacks [10, 49, 112] reduce the accuracy of the global model in general. As untargeted attacks are considered to be more severe (given they are harder to detect, cf. §5), we focus on untargeted attacks in the scope of this work.

2 Problem Statement

In this section, we define the precise problem of secure quantized aggregation for FL, which we address in our work. For doing so, we first introduce the necessary preliminaries on FL and quantization schemes, formalize the functionality we want to compute securely, and finally define our threat and system model for common (cross-device) FL scenarios.

2.1 Aggregation for Federated Learning

Google introduced federated learning (FL) as a distributed machine learning (ML) paradigm in 2016 [73, 89]. In FL, N data owners collaboratively train a ML model G with the help of a central aggregator S while keeping their input data locally private. In each training iteration t, the following three steps are executed:

1. The server S randomly selects n out of N available clients and provides the most recent global model G.
2. Each selected client C_i, i ∈ [n], sets its local model w_i^{t+1} = G_i and improves it using its local dataset D_i for E epochs (i.e., local optimization steps):

   \[ w_i^{t+1} \leftarrow w_i^{t+1} - \eta_c \frac{\partial L(w_i^{t+1}, B_{i,e})}{\partial w_i^{t+1}}, \]

   where L is a loss function, \eta_c is the clients’ learning rate, and B_{i,e} ⊆ D_i is a batch drawn from D_i in epoch e, where e ∈ [E]. After finishing the local training, C_i sends its local update w_i^{t+1} to S.
3. The server updates to a new global model G^{t+1} by combining all w_i^{t+1} with an aggregation rule \textit{f}_agg:

   \[ G^{t+1} \leftarrow G - \eta_s \cdot f_{agg}(w_1^{t+1}, \ldots, w_n^{t+1}), \]

   where \eta_s is the server’s learning rate. The most commonly used aggregation rule, which we also focus on, is FedAvg [89]. It averages the local updates as follows:

   \[ \text{FedAvg}(w_1^{t+1}, \ldots, w_n^{t+1}) = \frac{1}{n} \sum_{i=1}^{n} D_i \cdot w_i^{t+1} \]

   This process is repeated until some stopping criterion is met (e.g., a fixed number of training iterations or a certain accuracy is reached).

2.2 Stochastic Quantization

Quantization is a central building block in FL, where data transmission over the network is often a bottleneck. Thus, compressing the (thousands or millions of) gradients is essential to adhere to client bandwidth constraints, reducing training time, and allowing better inclusion and scalability. We now review desired properties and constructions of quantization schemes that will play a key role in our system design.

Unbiasedness. A well-known and desired design property of gradient compression techniques is being unbiased. That is, for an estimate \hat{w} of a gradient \dot{w} ∈ \mathbb{R}^d, being unbiased means that E[\hat{w}] = \dot{w}. Unbiasedness is desired because it guarantees that the mean’s estimation mean squared error (MSE) decays linearly with respect to the number of clients, which can be substantial in FL. In FL and other optimization techniques based on stochastic gradient descent (SGD) and its variants (e.g., [67, 83, 89]), the MSE measure (or normalized MSE, a.k.a. NMSE, cf. §3.2) is indeed the quantity of interest since it affects the convergence rate and often the final accuracy of the models. In fact, provable convergence rates for the non-convex compressed SGD-based algorithms have a linear dependence on the NMSE. Accordingly, to keep the estimates unbiased, modern quantization techniques employ stochastic quantization (SQ) and its variants to compress the gradients.

1-bit SQ. Our focus is on the appealing communication budget of a single bit for each gradient entry, resulting in a 32× compression ratio compared to regular 32-bit floating point entries. Indeed using 1-bit quantization has been the focus of many recent works concerning distributed and FL network efficiency (e.g., [17, 61, 111, 121, 123]). These works repeatedly demonstrated that a budget of 1-bit per coordinate is sufficient to achieve model accuracy that is competitive to that of a non-compressed baseline.

In particular, 1-bit SQ (i.e., SQ using two quantization values) can be done as follows: For a vector X with m dimensions, the client sends each coordinate as \sigma_i = \text{Bernoulli}(X_i \cdot \frac{s_X^{\max} - s_X^{\min}}{s_X^{\max} - s_X^{\min}}), where s_X^{\max} = \max(X) and s_X^{\min} = \min(X). It is then reconstructed by the receiver as \hat{X}_i = \sigma_i \cdot s_X^{\max} + \sigma_i \cdot (s_X^{\max} - s_X^{\min}). This simple technique results in an unbiased quantization as desired, i.e., E[\hat{X}_i] = E[\sigma_i \cdot s_X^{\max} + \sigma_i \cdot (s_X^{\max} - s_X^{\min})] = X_i. However, SQ-based techniques’ MSE is known to be sensitive to gradient coordinates’ range (i.e., the difference between the largest and smallest coordinates, s_X^{\max} − s_X^{\min}). Generally, and specifically in FL, neural network gradients are constantly increasing in size and can have significant differences in the magnitude of the different coordinates (e.g., several orders of magnitude).

Pre-processing via Random Rotations. To deal with possible limitations of vanilla SQ, recent state-of-the-art works suggest to randomly rotate the input vector prior to SQ [120]. That is, the clients and the aggregator draw rotation matrices according to some known distribution; the clients then send the quantization of the rotated vectors while the aggregator applies the inverse rotation on the estimated rotated vector.
Intuitively, the coordinates of a randomly rotated vector are identically distributed, and thus the expected difference between the coordinates is smaller, allowing for a more accurate quantization. For $n$ clients and a gradient with $m$ coordinates, this approach achieves a NMSE$^1$ of $O\left(\frac{\log m}{n}\right)$ using $O(m)$ bits, which asymptotically improves over the $O\left(\frac{m}{n}\right)$ NMSE bound of vanilla SQ. The computational complexity, on the other hand, is increased from $O(m)$ to $O(m\log m)$ when utilizing the randomized Hadamard transform for rotations.

**Pre-processing via Kashin’s Representation.** The rotation approach was recently improved using Kashin’s representation [28, 85, 109]. Roughly speaking, it allows representing an $m$-dimensional vector using a slightly larger vector with $\lambda \cdot m$ smaller coefficients ($\lambda > 1$). It can be shown that applying SQ to the Kashin coefficients allows for an optimal NMSE of $O\left(\frac{1}{\lambda}\right)$ using $O(\lambda \cdot m)$ bits. Compared with [120], Kashin’s representation yields a lower NMSE bound by a factor of $\log m$ at the cost of increasing the computational complexity by the same factor [12, 28].

**Other Approaches.** We acknowledge the existence of additional recent advances in gradient quantization (e.g., [12, 43, 122, 123]). However, these techniques do not allow aggregating gradients in their compressed form and therefore are less applicable to our framework.

**Linear SQ Techniques.** A key requirement of being able to perform secure aggregation **efficiently** is being able to aggregate client gradients in their compressed (i.e., quantized) form. A natural way to achieve this is using “global scales”. Namely, prior to quantization, clients securely learn the maximal and the minimal value across all gradients, i.e., each client uses $s^{\text{max}} = \max_x \{s^{\text{max}}_x\}$ and $s^{\text{min}} = \min_x \{s^{\text{min}}_x\}^2$. This way, summing the compressed vectors (i.e., their $\sigma$ indicators) and then decompressing the sum (i.e., scaling) is equal to decompressing each vector individually (i.e., scaling) and then computing their sum. Hereinafter, quantization techniques having this property are called “linear”.

The “global scales” approach is appealing due to its simplicity. However, it has several drawbacks: (1) it requires a preliminary communication stage, (2) it reveals the global extreme values (even if they are computed securely across all clients), and (3) the reconstruction error (i.e., the NMSE) is increased as it now depends on the extreme values across all round participants. Accordingly, we also consider a second approach where each client continues to use its own “local scales”. Since plainly using individual scales is not “linear”, i.e., it does not allow for aggregating the quantized gradients efficiently without decoding them, to realize this approach, we use a known approximation [14] and adapt it to our setting (cf. §4.1.3).

While in this work we focus on the mentioned vanilla SQ, SQ with random rotation as well as SQ with Kashin’s representation, our framework seamlessly supports any “linear” quantization scheme, namely, any quantization technique that allows for aggregation in a compressed form.

### 2.3 MPC for Secure Aggregation

Secure computation in the form of multi-party computation (MPC) allows a set of parties to privately evaluate any efficiently computable function on confidential inputs. This paradigm can be utilized to securely run the FedAvg aggregation algorithm [45, 51, 94, 97]: The set of selected FL clients uses information-theoretically secure additive secret sharing to distribute their sensitive inputs among a set of MPC servers, which resemble a distributed aggregator. The MPC servers securely add the received shares and reconstruct the public result from the resulting shares. In the next iteration, the public model is distributed to a new set of clients chosen at random, and the process is repeated until the desired accuracy is attained. A visualization of this outsourced MPC setting for secure aggregation is provided in Fig. 1.

In the remainder of this paper, we work towards a secure aggregation for FL using FedAvg on **quantized** inputs. In the following, we give our assumptions in terms of the threat model and refine the description of our system model.

**Semi-Honest Servers.** The MPC servers are assumed to be semi-honest. This means, they follow the protocol specification, but may try to learn additional information from the protocol messages that they receive. This well-established assumption [6, 21, 57, 63, 91, 97, 100] is motivated by the fact that companies who run FL with a secure aggregation scheme try to actively protect their clients’ data but want to make sure that someone who breaks into their systems cannot get plaintext access to the data that is being processed. Additionally, this assumption is justified as organizations usually cannot afford the reputational and monetary risk when being caught betraying their users’ trust. The protocols that we design provide this security guarantee in a dishonest majority setting, where data is protected even when an adversary $A$ corrupts more than 50% of the involved servers.

---

$^1$The normalized MSE is the mean’s estimate MSE normalized by the mean clients’ gradient squared norms.

$^2$This approach resembles “scaler sharing” in TernGrad [126].
Malicious Clients. On the contrary, we consider that some clients might behave maliciously by negatively affecting the quality of the global model with manipulated updates (known as poisoning attacks, cf. §5). This assumption stems from the fact that there are significantly less incentives for clients to behave honestly. Also, due to the sheer number of clients in cross-device FL, it is hard to verify their reputation.

Preprocessing Model. We focus mainly on MPC in the preprocessing model [13, 40–42, 70]. This means, we try to shift as much computation and communication as possible to a data-independent preprocessing or offline phase that can be executed at an arbitrary point in time before the actual computation. This gives several advantages. For example, in a cloud environment, service providers can exploit cheap spot instances. Furthermore, it guarantees faster results when the data-dependent online phase of the protocol is executed.

Shared Randomness. We assume that clients and MPC servers have access to a shared randomness source, e.g., by agreeing on a PRNG seed. Such a configuration has been widely employed in MPC protocols [6, 31, 52, 75, 92] as well as in ML systems [11, 120] to optimize communication.

2.4 Secure Quantized Aggregation

To introduce the problem of secure quantized aggregation, without loss of generality, we consider a simple stochastic binary quantization scheme to begin with. In this scheme, a m-dimensional vector of the form $X = \{x_1, \ldots, x_m\}$ comprising of ℓ-bit values will be quantized to obtain a triple $X_\sigma = (\tilde{\sigma}_X, s^\text{min}_X, s^\text{max}_X)$. Here, $\tilde{\sigma}_X$ is an m-dimensional binary vector with a zero at an arbitrary index indicating the value $s^\text{min}_X$ and a one indicating the value $s^\text{max}_X$. Also, $s^\text{min}_X$ and $s^\text{max}_X$ correspond to the minimum and maximum values in the vector $X$. With this binary quantization, the quantized value at the $i$th dimension, denoted by $\tilde{X}_i$, can be written as

$$\tilde{X}_i = s^\text{min}_X + \tilde{\sigma}_X \cdot (s^\text{max}_X - s^\text{min}_X).$$

See §2.2 for more concrete details of the scheme. Before going into the details of aggregation, we provide some of the basic notation that will be utilized throughout the paper.

Notation. $\mathbf{Y}_{\alpha \times \beta}$ denotes a matrix of dimension $\alpha \times \beta$ with $\mathbf{Y}_{ij}$ being the $i$th row and $\mathbf{Y}_{ij}$ being the $j$th column. An element in the $i$th row and $j$th column is denoted by $\mathbf{Y}_{ij}$. Also, $\text{Agg}-\mathbf{R}(\mathbf{Y}_{\alpha \times \beta})$ returns a row vector corresponding to an aggregate of the rows of $\mathbf{Y}$. Likewise, $\text{Agg}-\mathbf{C}(\mathbf{Y}_{\alpha \times \beta})$ returns an aggregate of the columns of $\mathbf{Y}$.

Given $\mathbf{U}_{\alpha \times \beta}$ and $\mathbf{V}_{\alpha \times 1}$, we use $\mathbf{U} \circ \mathbf{V}$ to denote the column-wise Hadamard product. Similarly, $\mathbf{U} \oplus \mathbf{V}$ denote the sum of two matrices in a column-wise fashion. Concretely, for $\mathbf{M}_{\alpha \times \beta} = \mathbf{U} \circ \mathbf{V}$ and $\mathbf{N}_{\alpha \times \beta} = \mathbf{U} \oplus \mathbf{V}$, we have $\mathbf{M}_i = \mathbf{U}_i \circ \mathbf{V}_i$ and $\mathbf{N}_i = \mathbf{U}_i + \mathbf{V}_i$, where $i \in [\alpha], j \in [\beta]$.

Quantized Aggregation. To perform aggregation on quantized inputs, a set of $n$ clients first locally prepares their quantized triples, $(\tilde{\sigma}_X, s^\text{min}_X, s^\text{max}_X)$, corresponding to their locally trained ML model updates (i.e., gradients) and submits to a parameter server for aggregation. Let $m$ be the dimension of the underlying ML model. The quantized triples can then be consolidated to a matrix triple of the form $(\mathbf{B}_{n \times m}, \mathbf{U}_{n \times 1}, \mathbf{V}_{n \times 1})$. Here, $\mathbf{B}$ is a binary matrix that corresponds to the $\tilde{\sigma}_X$ vector of the clients. Similarly, $\mathbf{U}$ and $\mathbf{V}$ correspond to the $s^\text{min}_X$ and $s^\text{max}_X$ values of the above-mentioned triple. The aggregate of the quantized inputs is defined as

$$X_{1 \times m} = \text{Agg}-\mathbf{R}(\mathbf{U}_{n \times 1} \oplus \mathbf{B}_{n \times m} \circ (\mathbf{V}_{n \times 1} - \mathbf{U}_{n \times 1})). \quad (5)$$

Ideal Functionality. To address the aforementioned problem using secure computation in the form of MPC, we model it as an ideal functionality $\mathcal{F}_{\text{SecAgg}}$, which is provided in Fig. 2. We consider a set of $\tau$ servers to which the clients secretly share their quantized updates and the goal is to compute the aggregate of the inputs as shown in Eq. 5. Let $\langle \cdot \rangle$ denote the underlying secret sharing scheme. Looking ahead, we will use linear secret sharing techniques for MPC, in which linear operations such as addition and subtraction are local.

As a result, we will concentrate on efficiently computing the column-wise Hadamard product.

![Figure 2: Ideal functionality for semi-honest secure quantized aggregation for linear stochastic binary quantization.](image-url)

3 Basic Approach

As a warm-up, we start with solving a simplified secure quantized aggregation problem. For this, we assume that all the clients use the same set of scales for quantization, denoted by $s^\text{min}_G$ and $s^\text{max}_G$. In this case, it is sufficient to compute

$$\tilde{X}_{1 \times m} = s^\text{min}_G \oplus \text{Agg}-\mathbf{R}(\mathbf{B}_{n \times m}) \circ (s^\text{max}_G - s^\text{min}_G) \quad (6)$$

as the aggregation result. This corresponds to the “global scales” approach discussed in §2.2. When $s^\text{min}_G = 0$ and $s^\text{max}_G = 1$, this can also be viewed as an instance of privacy-preserving aggregate statistics computation, as demonstrated in the works of Prio [36] and Prio+ [2].
The Problem of Share Conversion. For all of the operations required to compute Eq. (6) (i.e., addition, subtraction, and multiplication), there are well-known and efficient MPC protocols available. However, one difficulty arises from the fact that for minimal communication overhead, in a simple additive secret sharing scheme, clients will share the binary matrix \( \tilde{B}_{m \times n} \) over \( \mathbb{Z}_2 \), meaning each bit \( b \) will be split into \( \tau \) Boolean shares \( b_1, \ldots, b_\tau \) such that \( b = b_1 \oplus \cdots \oplus b_\tau \). This way, for each bit in \( \tilde{B}_{m \times n} \), each client only has to transfer a single bit to each MPC server. (We will later discuss how the client-server communication in our setting can be reduced from \( \tau \) bits to a single bit.) Unfortunately, to perform the operation \( \text{Agg-R} \) over secret-shared values, the MPC servers first must convert these Boolean-shared bits into arithmetic shares over a larger ring \( \mathbb{Z}_{2^\ell} \). Prior works solve this problem either by using a naive and costly standard conversion technique [2] or require the clients to provide inputs already in an arithmetically shared form [14], which we want to avoid due to the targeted cross-device FL setting where low client-server communication overhead is crucial.

In this section, we first discuss the naive standard conversion technique and focus on how to optimize the communication and computation costs using an approximate method. We later describe our full system considering private local scales and further optimizations in §4.

### 3.1 Secure Bit Aggregation

This section discusses methods for computing as well as estimating a bit’s value and generating its arithmetic sharing from its Boolean shares. To begin with, consider the case of bit \( b \) represented using two Boolean shares \( b_1, b_2 \in \{0, 1\} \), such that \( b = b_1 \oplus b_2 \). Note that when embedding \( b_1 \) and \( b_2 \) in a larger field/ring \(^3\), it holds that \( b = b_1 + b_2 \). Similarly, for \( b = b_1 \oplus b_2 \oplus b_3 \), \( b = b_1 + b_2 + b_3 \). This concept can be generalized to an arbitrary number of shares, denoted by \( q \), as discussed below. For \( b = \oplus_{i=1}^q b_i \), let \( Q = \{b_i\}_{i \in [q]} \) denote the set of all \( q \) shares of \( b \), and \( b \) the arithmetic equivalent of the share \( b_i \). Let \( 2^Q \) be the powerset of \( Q \) and \( Q^{[\ell]} \) the set of all size-\( \ell \) subsets in \( 2^Q \), that is, \( 2^Q = \sum_{i=0}^q Q^{[l]} \). The arithmetic equivalent of \( b \), denoted by \( \tilde{b} \), is given as

\[
\tilde{b} = \sum_{\{b_i\} \in Q^{[\ell]}} \prod_{b \in Q^{[\ell]}} b_i = \sum_{\ell=1}^q (-2)^{q-\ell} \prod_{b \in Q^{[\ell]}} b_i
\]

For the analysis, we divide Eq. (7) into three terms: Sum (term\(_s\)), Middle (term\(_m\)), and Product (term\(_p\)) as shown in Eq. (8) below. (Note that \( Q^{[q]} = Q \)).

\[
b = \sum_{\{b_i\} \in Q^{[\ell]}} \prod_{b \in Q^{[\ell]}} b_i + \sum_{\ell=2}^q (-2)^{q-\ell} \prod_{\{b_i\} \in Q^{[\ell]}} \prod_{b \in Q^{[\ell]}} b_i + (-2)^{q-1} \prod_{b \in Q} b_i
\]

### Our Approach

Performing this conversion in MPC requires many additions and multiplications. While linear operations like additions can be calculated for “free” in most MPC protocols, non-linear operations such as multiplications require some form of communication between the MPC servers. Hence, computing the middle term is costly, especially when there is a large number of shares involved. Instead, we investigate a novel approximation approach: To approximate \( \tilde{b} \) in Eq. (8), we keep the sum and product terms unchanged and replace only the middle term. At a high level, the idea is to replace term\(_m\) in Eq. (8) with its expected value \( E[\text{term}_m] \) (that depends on \( q \)) such that the approximate value of \( \tilde{b} \), denoted by \( \tilde{b}_a \), retains \( E[\tilde{b}_a] = b \). The expectation of term\(_m\) and term\(_p\) in Eq. (8) is first calculated, and \( E[\text{term}_m] \) is inferred using the fact that \( E[\tilde{b}_a] = b \). This analysis is summarised in Lem. 3.1.

#### Lemma 3.1 (Expected Values)

Given a bit \( b = \oplus_{i=1}^q b_i \) and \( b = \text{term}_s + \text{term}_m + \text{term}_p \) with

\[
\text{term}_s = \sum_{\{b_i\} \in Q^{[\ell]}} \prod_{b \in Q^{[\ell]}} b_i, \quad \text{term}_m = \sum_{\ell=2}^q (-2)^{q-\ell} \prod_{\{b_i\} \in Q^{[\ell]}} \prod_{b \in Q^{[\ell]}} b_i,
\]

we have \( E[\text{term}_s | b] = q/2 \), \( E[\text{term}_m | b] = (q-1) \) mod 2 - \( q/2 \), and \( E[\text{term}_p | b] = b - (q-1) \) mod 2.

#### Proof

For the analysis, we use the truth table of \( b \), denoted by \( T_b \), which has \( 2^q \) rows. Half of the rows in \( T_b \) correspond to \( b = 0 \), while the other half correspond to \( b = 1 \). The truth table for three shares (\( q = 3 \)) is given in Tab. 1 as a reference.

| \( b \) | \( b_1 \) | \( b_2 \) | \( b_3 \) | \( \text{term}_s \) | \( \text{term}_m \) | \( \text{term}_p \) | \( \tilde{b} \) |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| 1     | 0     | 0     | 0     | 1     | 0     | 0     | 1     |
| 1     | 0     | 0     | 0     | 0     | 1     | 0     | 0     |
| 0     | 0     | 1     | 0     | 2     | 0     | 0     | 2     |
| 1     | 1     | 0     | 0     | 1     | 1     | 0     | 1     |
| 0     | 1     | 0     | 0     | 1     | 1     | 0     | 1     |
| 0     | 1     | 0     | 0     | 2     | 2     | 0     | 0     |
| 0     | 1     | 0     | 0     | 0     | 0     | 0     | 0     |
| 1     | 1     | 1     | 1     | 3     | -6    | 4     | 1     |

Table 1: Truth table for \( b = b_1 + b_2 + b_3 \). The rows corresponding to \( b = 0 \) are highlighted. \( \tilde{b} \) denotes the arithmetic equivalent of \( b \).

#### Sum Term (term\(_s\))

For each row of the form \( \{b_1, \ldots, b_q\} \) in \( T_b \), term\(_s\) equals \( b_1 + \cdots + b_q \), which can be interpreted as the number of \( b_i \)'s selected out of the \( q \) possible. Furthermore, there are a total of \( \binom{q}{\ell} \) rows with sums equal to \( \ell \), with \( k \) being...
odd corresponding to the row for \( b = 1 \) and \( k \) being even
corresponding to the row for \( b = 0 \). As a result, given \( b = 0 \),
the expectation of the sum term can be calculated as the product of \( 1/2^q - 1 \)
(corresponding to rows in \( T_b \), with \( b = 0 \)) and
the sum of terms of the form \( k \cdot \binom{n}{k} \) with \( k \) being even.
Using Lem. B.1 in App. B.1, we get
\[
E[\text{term}_s | (b = 0)] = \frac{1}{2^q - 1} \sum_{k=0}^{\binom{q}{2}} 2k \binom{q}{2k} = \frac{1}{2^q - 1} \cdot q \cdot 2^{q-2} = q/2.
\]

Similarly, we obtain \( E[\text{term}_s | (b = 1)] = q/2 \). To summarize,
we have \( E[\text{term}_s | b] = q/2 \).

**Product Term** \((\text{term}_p)\): The product of all the \( q \) shares will
be 1 only if all the shares are 1, otherwise it will be 0. Moreover,
almost all shares of \( b \) being 1 correspond to \( b = 1 \) if \( q \) is odd,
and \( b = 0 \) otherwise. Now, when \( q \) is odd then \( \text{term}_p = (-2)^{q-1} \) with probability \( \frac{q}{2^q} \)
(when all the shares of \( b \) are 1, given that at least one share is 1 as we are in the case \( b = 1 \),
and 0 otherwise. In this case, we can write
\[
E[\text{term}_p | (b = 0 \land q \text{ is odd})] = \frac{1}{2^q - 1} \cdot 0 = 0
\]
\[
E[\text{term}_p | (b = 1 \land q \text{ is odd})] = \frac{1}{2^q - 1} \cdot (-2)^{q-1} = 1
\]
Similarly, the case for even \( q \) can be written as
\[
E[\text{term}_p | (b = 0 \land q \text{ is even})] = \frac{1}{2^q - 1} \cdot (-2)^{q-1} = 0
\]
\[
E[\text{term}_p | (b = 1 \land q \text{ is even})] = \frac{1}{2^q - 1} \cdot 0 = 0
\]
The above observation can be summarized as \( E[\text{term}_p | b] = b - (q-1) \mod 2 \).

**Middle Term** \((\text{term}_m)\): Given \( E[b] = b \), \( E[\text{term}_s | b] \)
and \( E[\text{term}_p | b] \), the expectation of \( \text{term}_m \) can be calculated as
\[
E[\text{term}_m | b] = E[b] - E[\text{term}_s | b] - E[\text{term}_p | b] = b - q/2 - (b - (q-1) \mod 2) = (q-1) \mod 2 - q/2.
\]

This concludes the proof of Lem. 3.1.

**Our Approximation.** With the aforementioned observations,
the approximate arithmetic equivalent of \( b \), denoted by \( \hat{b} \), is as follows:
\[
\hat{b} = \sum_{b_e \in Q} b_e \left( \binom{q-1}{2} \mod 2 - \frac{q}{2} + (-2)^{q-1} \prod_{b \in Q} \hat{b}_e \right) \quad (9)
\]
While \( \text{term}_s \), is kept because it only involves linear operations
on the shares of \( b \) (which are free in MPC for any linear secret sharing scheme),
we observe that \( \text{term}_p \) is required to keep the expected values for \( b = 0 \) and \( b = 1 \) different. This is evident from Lem. 3.1
where \( E[\text{term}_p | b] \) is the only term that depends on \( b \).

In general, if a term that depends on all the \( q \) shares of \( b \) is
missing from the approximation, we get \( E[b = 0] = E[b = 1] \).
The intuition is that only such a term can differentiate between
\( b = 0 \) and \( b = 1 \), while all other terms will be symmetrically distributed. For instance, consider \( q = 3 \) and let \( b = c_1b_1 + c_2b_2 + c_3b_3 + c_4b_1b_2 + c_5b_2b_3 + c_6b_1b_3 + c_7 \) for some random combiners \( c_i \in \mathbb{Z}_{2^q} \) and \( i \in \{7\} \). Using the truth table \( T_b \)
given in Tab. 1, it is easy to verify that
\[
E[b = 0] = E[b = 1] = \frac{1}{4} \cdot (2c_1 + 2c_2 + 2c_3 + c_4 + c_5 + c_6 + 4c_7) \quad (10)
\]
This argument can be generalized to any value of \( q \).

**Claim 3.2.** The approximate arithmetic equivalent \( \hat{b} \)
in Eq. (9) preserves the expectation of the exact bit \( b \) in Eq. (7),
i.e., \( E[\hat{b} = 0] = 0 \) and \( E[\hat{b} = 1] = 1 \).

**Proof.** The proof is straightforward as we replace the middle
term \((\text{term}_m)\) in Eq. (7) with its expected value \( \text{term}_m^\text{ag} \).

In App. B.2.1, we provide an analysis of the efficiency gains
achieved by our approximation, showing that the number of
cross terms that must be securely computed using MPC is
reduced exponentially in the number of shares \( q \).

### 3.2 Accuracy Evaluation

In §3.1, we showed that our approximate bit conversion preserves
the expectation of the exact bits. However, we also want to understand the concrete accuracy impact on the aggregation result due to the increased variance. For this, we simulate the aggregation of random vectors \( \vec{v} \) with dimension \( d \) drawn from a \((0,1)\)-log-normal distribution.\(^4\) Then, we measure the normalized mean square error (NMSE) when comparing the averaged aggregation result \( \text{agg} \) computed on secret-shared and quantized inputs to the plain averaged aggregation result due to the increased variance. For this, we measure
\[
\text{NMSE} = \frac{\| \vec{a} - \vec{b} \|_2^2}{\| \vec{b} \|_2^2},
\]
where \( \vec{a} \) is computed using various linear quantization schemes
with global scales (1) with an exact bit-to-arithmetic conversion and (2) with our approximation enabled. Our results shown in Fig. 3 are the average of 10 trials for each experiment with \( q = 3 \) shares (representing a three-server dishonest majority setting using the masked evaluation technique that we will introduce in §4). Consistently, we observe that applying our approximation increases the NMSE by about three orders of magnitude for stochastic quantization without rotation, and by about one and a half orders of magnitude for rotation-based algorithms. In Fig. 17 in App. B.2.3, we additionally provide an evaluation for the case of local scales, which we will use in our final system design, showing that the error can be significantly reduced. Furthermore, as we will show in §4.2, the error is still so small that the impact on the accuracy in common FL settings is negligible.

\(^4\)We use this distribution for preliminary measurements as it was commonly observed in neural network gradients (e.g., [35]).
4 Our System: ScionFL

In this section, we present our system ScionFL in detail from an MPC standpoint. We present the underlying sharing semantics, client interaction with MPC servers, and methods for performing secure aggregation of client updates. We begin by introducing the masked evaluation technique, which is used to improve our system’s real-time performance.

**Masked Evaluation.** In our MPC protocols, we use the masked evaluation technique [16, 32, 55, 75, 88, 119], which enables the costly data-independent calculations to be completed in a preprocessing phase, thus enabling a fast and efficient data-dependent online phase (cf. §2.3). In this model, the secret-share of every element \( v \in \mathbb{Z}_q \), denoted by \( \langle v \rangle \), is associated with two values: a random mask \( \lambda_v \in \mathbb{Z}_q \) and a masked value \( m_v \in \mathbb{Z}_q \) such that \( v = m_v + \lambda_v \). While \( \lambda_v \) is split and distributed as \( q \) shares similar to the general case discussed in §3.1, \( m_v \) is given to all MPC servers. Since \( \lambda_v \) is random and independent of \( v \), all operations involving \( \lambda_v \) can be computed when the preprocessing phase is complete.

**Masked Bit Conversion and Injection.** With the masked evaluation technique, the arithmetic equivalent \( \bar{b} \) for a bit \( b = m_b \oplus \lambda_b \) can be obtained as

\[ \bar{b} = m_b \oplus \lambda_b = M_b + (1 - 2m_b) \cdot \Lambda_b, \]  

where \( M_b \) and \( \Lambda_b \) denote the arithmetic equivalents of \( m_b \) and \( \lambda_b \), respectively. Note that \( \Lambda_b \) can be computed from \( \lambda_b \) during the preprocessing phase using techniques discussed in §3.1, including our novel approximation from §3.1. Similarly, given \( s = M_s + \Lambda_s \), the value \( b \cdot s \) can be obtained as

\[ b \cdot s = (M_b + (1 - 2m_b) \cdot \Lambda_b) \cdot (M_s + \Lambda_s) = M_b M_s + M_b \Lambda_s + (1 - 2m_b) \cdot (\Lambda_b M_s + \Lambda_b \Lambda_s). \]  

As shown in Eq. (13), the servers need to additionally compute \( \Lambda_b \Lambda_s \) in the preprocessing phase.

---

**Client Interaction.** Before going into the details of aggregation among the \( r \) MPC servers, we discuss input sharing and the reconstruction of the aggregated vector for clients.

To generate the \( \langle \cdot \rangle \)-shares of a value \( v \in \mathbb{Z}_q \) owned by client \( C \), it first non-interactively computes the additive shares of the mask \( \lambda_v \) using the shared randomness setup discussed in §2.3. The masked value is then computed as \( m_v = v - \lambda_v \) and sent to a single designated MPC server, say \( S_1 \). The input sharing is completed when \( S_1 \) sends \( m_v \) to all remaining MPC servers. For Boolean values (i.e., \( \mathbb{Z}_2 \)) the procedure is similar except that addition/subtraction is replaced with XOR and multiplication with AND. We use \( \langle \cdot \rangle^B \) to denote the secret sharing over the domain \( \mathbb{Z}_2 \).

After the servers have received all inputs in \( \langle \cdot \rangle \)-shared form, they instantiate the \( f_{SecAgg} \) functionality specified in Fig. 2 and obtain the aggregated vectors in \( \langle \cdot \rangle \)-shared form. Recall from §2.1 that the values to be aggregated in our case correspond to FL gradients, and the aggregated result can also be made public. As a result, the servers reconstruct the aggregated result towards a chosen server, say \( S_1 \), which updates the global model according to Eq. (3). In the next iteration, \( S_1 \) distributes the updated global model to a fresh set of clients, and the process is repeated.

From a client’s perspective, it is only interacting with one server (apart from a one-time shared-randomness setup), as it is the case in a privacy-free variant in which clients communicate with a single parameter server for aggregation [89].

**MPC Protocols.** We provide the full MPC protocols for inner product computation, multiplication, and bit-to-arithmetic conversion in App. B.1.2, which are utilized in a black-box fashion in this section.

4.1 MPC-based Aggregation

We now discuss two approaches for instantiating \( f_{SecAgg} \) using MPC protocols that operate on secret-shared values. Recall from Eq. (5) that the MPC servers for the aggregation of quantized values possess \( \langle \cdot \rangle \)-shares of matrices \( \bar{U}_{n \times 1} \) and \( \bar{V}_{n \times 1} \) along with the \( \langle \cdot \rangle^B \)-shares of \( \bar{B}_{n \times m} \).

4.1.1 Approach I

A naive instantiation of \( f_{SecAgg} \) would be to have the servers convert binary shares of the matrix \( \bar{B} \) to their arithmetic shares first, as in Prio+ [2]. This is possible in MPC with a bit-to-arithmetic conversion protocol [2, 75, 92, 102], denoted by \( \Pi_{BitA} \), which computes the arithmetic shares of \( b \in \mathbb{Z}_2 \) from its Boolean sharing. Once the arithmetic shares are generated, the servers use an instance of an inner-product protocol, denoted by \( \Pi_{IP} \), on each of the \( m \) columns of matrix \( \bar{B} \) with the column vector \( \bar{V}_{n \times 1} - \bar{U}_{n \times 1} \) computed locally to obtain the row aggregate. They complete the protocol by adding a row aggregate of \( \bar{U} \) to each column of the resultant matrix computed in the previous step. The formal protocol \( \Pi_{SecAgg}^{II} \) is given in Fig. 4.
We use the bit injection protocol [74, 92, 102], denoted where the inner product between bits and scales:

inner product between bits and scales:

secret sharing for FL. Unlike our work, which investigates the SIGNSGD compression technique of [17] with additive quantization, we perform one secure multiplication per coordinate, with the other operations being linear and free in our MPC protocol. As a result, we can utilize linear quantization schemes with local scales at the cost of global scales (ignoring the overhead for global scales to securely determine $s_{x_{min}}^g$ and $s_{y_{min}}^g$ across all participants). The formal protocol $\Pi_{\secagg}^I$ utilizing SepAgg appears in Fig. 6.

4.1.2 Approach II

We use the bit injection protocol [74, 92, 102], denoted by $\Pi_{\bita}$, which computes the arithmetic sharing of $b \cdot s$ given the Boolean sharing of $b \in \mathbb{Z}_2$ and the arithmetic sharing of $s \in \mathbb{Z}_2$. Given $\langle \mathbf{M}_{n \times 1} \rangle$ and $\langle \mathbf{N}_{n \times 1} \rangle$, the high-level idea is to effectively combine the $\Pi_{\bita}$ and $\Pi_{\bit}$ protocol to a slightly modified instance of $\Pi_{\bita}$ that directly computes the sum $\langle \sum_{i=1}^{n} \mathbf{M} \cdot \mathbf{N} \rangle$ instead of the individual positions. This can be achieved following Eq. (13) and the details appear in Fig. 15 in §B.1.2. One significant advantage of this technique is that online communication is no longer dependent on the number of clients $n$. $\Pi_{\secagg}^I$ denotes the resulting protocol and the formal details are given in Fig. 5.

4.1.3 Approach III using SepAgg

In a closely related work [14], the authors combine the SIGNSGD compression technique of [17] with additive secret sharing for FL. Unlike our work, which investigates secure aggregation using various linear quantization algorithms in a cross-device environment, they aim for a cross-silo setting in which clients distribute arithmetically shared values to servers rather than single bits. In terms of client-server communication, this indicates an overhead of factor $\log_2 n$, where $n$ is the number of involved participants each round.

However, the authors of [14] introduce an interesting approximation called “SepAgg” for computing the averaged inner product between bits and scales:

$$\frac{1}{n} \sum_{i=1}^{n} \mathbf{s}_X \cdot \mathbf{d}_X \approx \frac{1}{n^2} \left( \sum_{i=1}^{n} \mathbf{s}_X \right) \left( \sum_{i=1}^{n} \mathbf{d}_X \right). \quad (14)$$

We adopt the SepAgg method to our setting to compute $\text{agg}(\dot{\mathbf{R}} \cdot (\mathbf{V} − \dot{\mathbf{U}}))$ in Eq. (5). In particular, we aggregate the matrices $\mathbf{B}$ and $(\mathbf{V} − \dot{\mathbf{U}})$ independently and then perform one secure multiplication per coordinate, with the other operations being linear and free in our MPC protocol. As a result, we can utilize linear quantization schemes with local scales at the cost of global scales (ignoring the overhead for global scales to securely determine $s_{x_{\text{min}}}^g$ and $s_{y_{\text{min}}}^g$ across all participants). The formal protocol $\Pi_{\secagg}^I$ utilizing SepAgg appears in Fig. 6.

Accuracy Evaluation. We next provide strong empirical evidence that applying SepAgg for SQ with preprocessing preserves the linear NMSE decay with respect to the number of clients (i.e., unbiased estimates). For this, we run a simulation similar to the one described in §3.2. Here, we compare the NMSE computed as in Eq. (11) for an aggregation $\text{agg}$ (1) a regular dot product between converted bits and scales and (2) with SepAgg. Note that here we do not apply our approximate conversion presented in §3.1 to study the effect of SepAgg in isolation. Our results shown in Fig. 7 are the average of 10 trials for each experiment. While for smaller vector dimensions, we observe a visible effect for SQ without preprocessing, there is only a minor difference for the other two quantization schemes, and sometimes the NMSE for SepAgg is even smaller than for the exact computation.

Formally proving that applying SepAgg after different preprocessing techniques, such as random rotation and Kashin's method, preserves the linear decay requires a more involved analysis. However, we show in Fig. 7 that SepAgg results in smaller NMSE across all dimensions and number of clients. We also observe that SepAgg has a smaller standard deviation compared to vanilla SQ. This suggests that SepAgg is a more robust approach to quantization in FL.
representation, results in an unbiased aggregation is a significant theoretical challenge. We leave this intriguing endeavor for future work.

**Communication Costs.** We provide a detailed analysis of theoretical and concrete communication costs for each of our three approaches outlined in §4.1.1 to §4.1.3 in App. B.2.4.

### 4.2 Performance Evaluation

We implement an extensive end-to-end FL evaluation and MPC simulation framework. We first describe our implementation, then the parameters used for our accuracy evaluation, and finally the obtained results.

**Implementation.** Our implementation is written in Python based on the PyTorch ML framework. Hence, it supports multi-GPU acceleration, also for the MPC simulation. A subset of this framework was already used for measuring the accuracy of our approximate bit conversion (cf. §3.2) and SepAgg (cf. §4.1.3), and we will describe extensions in §5.1.2 to incorporate evaluations of poisoning attacks and defenses.

On a high-level, our frameworks provide a simple CLI to run FL training tasks and observe the resulting training as well as test accuracy. Upon execution, the framework distributes training data among the specified number of virtual clients that locally perform optimization. The server(s) perform aggregation using FedAvg. When the MPC simulation is enabled, the clients’ input will be secret-shared before aggregation and the protocol described in §4.1.3 will be locally executed. Note that here our goal is not to assess the runtime performance of the MPC protocol but rather precisely measure the impact on the accuracy. Our implementation supports all exact and approximate secure aggregation variants described in this paper.

**Parameters.** We evaluate the accuracy on the following standard FL tasks for image classification: training (1) LeNet on MNIST [80] for 1000 rounds and (2) ResNet9 on CIFAR-10 [77] for 8000 rounds. For all tasks, we set a client batch size of 8, a learning rate of 0.05, and perform 5 local client train steps per round. For MNIST, we run training using \( N \in \{200,1000,5000\} \) clients and choose 10% of the clients at random per round. Due to the memory constraints of our system (that simulates all clients at once), we restrict training for CIFAR10 to \( N = 1000 \) clients and select \( n = 40 \) per round. As we observed a significant loss in accuracy for plain SQ in our accuracy evaluation for approximate bit conversion as well as SepAgg (cf. §3.2), we focus our evaluation on more accurate linear quantization schemes, i.e., HSQ and KSQ. For the MPC simulation of our approximate secure aggregation following Approach-III (cf. Fig. 6), we choose a three-server dishonest majority setting.

**Results.** The results for the MNIST/LeNet training are given in Fig. 8. Validation accuracy for our approximate version converges to almost the same final accuracy as the insecure exact aggregation. Specifically, in the final round of training, the difference between the two is diminished to 0.77% and 0.33% for HSQ and KSQ for \( N = 5000 \), respectively. Similar results can be observed for CIFAR10/ResNet9 in Fig. 9. However, here the difference between the exact and approximate version with 3.14% in the final round for KSQ is higher. This gap is expected due to the significantly lower number of clients per round, for which our approximate bit conversion and SepAgg technique result in a comparatively high NMSE over the baseline (cf. Figs. 3 and 7). We expect this effect to vanish for a real cross-device setting with thousands of participants per round (due to the demonstrated linear decay of the NMSE when increasing \( n \)), which we unfortunately cannot simulate with complex network architectures due to hardware limitations. Additionally, one may use a hybrid approach, where training is conducted with the approximate version for the initial rounds until a baseline accuracy is reached, whereas secure exact training (potentially including only the SepAgg [14] approximation but not our approximate bit-to-arithmetic conversion) is used for fine tuning up to the desired target accuracy.

In Tab. 2, we additionally compare the exact inter-server MPC communication cost for a naive MPC implementation of the exact computation to our optimized approximate version including SepAgg. As we can see, we improve the offline communication by factor \( \approx 15 \times \). For the online communication, we can see a wide range of improvement factors from \( 22 \times \) to \( 547 \times \) for MNIST with \( n = 500 \). This highlights the positive impact when utilizing the SepAgg approach for aggregating an increasingly large number of rows,

![Figure 8: Validation accuracy for training LeNet on the MNIST data set for \( N \in \{200,1000,5000\} \) clients when selecting 10% of the clients at random per round (\( n \)) for SQ with Hadamard (HSQ, left) and with Kashin’s representation (KSQ, right); “exact” denotes the insecure baseline, “approx” the simulation of our MPC-based approximate secure aggregation including SepAgg (cf. Fig. 6).](image-url)
non-interactively. Note that there are slight differences in communication overhead for HSQ and KSQ. This is because for an efficient GPU-friendly implementation of the randomized Hadamard transform, which we use for both rotating the gradients in HSQ and for calculating Kashin’s coefficients in KSQ, we require that the gradients’ size are a power of 2. In App. B.1.1, we detail how we can minimize the resulting overhead by dividing the gradients into chunks, and we also give the exact number of bits per gradient that we assume in our calculations for each algorithm.

![Validation Accuracy](image)

**Figure 9:** Validation accuracy for training ResNet9 on the CIFAR10 data set for \( N = 1000 \) clients with random \( n = 40 \) selected per round for quantization techniques and protocols as in Fig. 8.

For a more fine-grained analysis, Table 2 shows the inter-server communication per round for our benchmarks for different numbers of clients \( n \) in MiB.

| Benchmark | \( n \) | Method | Naive Exact (cf. Fig. 4) | Our Approx. (cf. Fig. 6) |
|-----------|-------|--------|--------------------------|------------------------|
| MNIST/LeNet | 20 | HSQ | 572.89 | 11.47 | 58.26 | 0.52 |
| MNIST/LeNet | 100 | HSQ | 2864.46 | 57.34 | 187.49 | 0.52 |
| MNIST/LeNet | 500 | HSQ | 14322.28 | 286.72 | 833.64 | 0.52 |
| CIFAR10/ResNet9 | 40 | HSQ | 87079.45 | 1743.26 | 6883.13 | 39.85 |
| | | KSQ | 100828.84 | 2018.51 | 7969.94 | 46.14 |

Table 2: Inter-server communication per round for our benchmarks for different numbers of clients \( n \) in MiB.

## 5 Defending Untargeted Poisoning Attacks

Beyond data privacy threats, FL was also shown to be vulnerable to manipulations affecting the inference service quality a model is trained for. Recall from Eq. (1) in §2.1 that in FL, each training iteration \( t \) consists of participating clients submitting their locally computed gradients to a central aggregator, who combines them to obtain the global model for the following iteration. An adversary \( \mathcal{A} \) who controls a fraction of the participating clients, on the other hand, can alter their local models or model updates and send these malformed updates for aggregation, affecting the characteristics of the final global model [9, 19, 49, 112, 113, 127]. This highlights the need for effective “defense measures” capable of thwarting such attacks. There are a plethora of such attacks and defenses in the FL literature and, we give an overview of the state-of-the-art works that are closely related to ours in App. §A.5.

We focus on defending against _untargeted poisoning attacks_, in which the attacker attempts to damage the trained model’s performance for a large number of test inputs, typically resulting in a final global model with a high error rate [10, 49, 112]. These types of attacks pose a significant threat to the deployment of FL for two reasons: (1) Untargeted attacks are particularly difficult to detect because, ignorant of the attack, service providers are unaware that they could have achieved a greater accuracy. (2) Even a minor drop in accuracy can cause enormous (competitive) damage [113].

### Min-Max Attack [112]

Most proposed untargeted poisoning attacks on FL use the (unrealistic) assumption that the adversary \( \mathcal{A} \) is aware of either the aggregation rule [49] or all benign updates [10]. However, the _Min-Max_ attack proposed by Shejwalkar and Houmansadr [112] defies this assumption and constitutes the state-of-the-art attack. This attack prevents the manipulations from being detected by allowing the adversary to compute representative benign updates using some clean training data and then using those in the attack to limit the maximum distance of the manipulated update to any other update by the maximum distances of any two benign updates. This ensures that the malicious gradients are sufficiently similar to the set of benign gradients. We refer to [112, §IV] for more specifics on the attack.

In addition to removing the assumptions about the adversary’s knowledge, the authors experimentally demonstrate that this attack outperforms the then-state-of-the-art poisoning attack [10] for almost all tested datasets. However, since all of the benchmarks in [10, 112] were performed on FL schemes without quantization, the impact of the _Min-Max_ attack on quantized FL schemes is unclear. Hence, we initially test the attack’s effectiveness in our framework using the open-sourced code\(^6\) as the baseline. As will be discussed in §5.1.2, we observe that the attacks are effective even in the context of quantization. As a result, we design a defense called ScionFL-Aura against untargeted poisoning attacks, which we detail next.

### 5.1 Our Defense: ScionFL-Aura

From an intuitive standpoint, the adversary in an untargeted poisoning attack seeks to manipulate the global update with malicious updates to deviate it as much as possible from the result of an ideal benign training while evading potentially deployed detection mechanisms. This baseline observation was also used by earlier proposed defense mechanisms [97, 112], however, those cannot be combined trivially with ScionFL without having to de-quantize all updates and running expensive secure computation machinery.

In this section, we first outline the general design of ScionFL-Aura and show its effectiveness against the _Min-Max_ attack [112]. Then, in §5.1.3, we outline how to efficiently instantiate it in an MPC-friendly manner to reduce communication overhead over the baseline approach.

\(^6\)https://github.com/vrt1shjwlkr/NDSS21-Model-Poisoning
5.1.1 Approach

ScionFL-Aura uses a hybrid approach by combining ideas from existing FL defenses based on $L_2$-norm [9, 97, 118] and cosine similarity [29, 97]. Several works like [97] compute these metrics for each pair of clients, resulting in expensive computations. In contrast, we first aggregate all updates, including the poisoned ones, to produce the vector $\vec{X}_{agg}$, which we then utilize as the reference.

At a high level, scaling based on the $L_2$-norm of the gradient vectors is used at first to bound the impact of malicious contributions that are potentially overlooked (i.e., not filtered) in later stages. In a second step, local updates that significantly deviate from the average update direction are considered to be manipulated and, thus, excluded. Concretely, ScionFL-Aura consists of the following steps:

1. **$L_2$-norm-based Scaling.** In this step, the $L_2$-norm of each gradient vector is compared against a public threshold multiplied with the average of the $L_2$-norms. Let $\mu_{th}$ denote the threshold and $L_{avg}^2$ denote the average of the $L_2$-norms across all clients. If $L^2 > \mu_{th} \cdot L_{avg}^2$ for a gradient vector $\vec{X}$, the vector is scaled by a factor of $\mu_{th} \cdot L_{avg}^2 / L^2$. This ensures that no gradient has an $L_2$-norm greater than $\mu_{th} \cdot L_{avg}^2$.

2. **Cosine-distance-based Filtering.** This step involves computing the cosine distance of each gradient from the reference vector $\vec{X}_{agg}$ discussed above. Following that, another round of aggregation is performed on the updated vectors, but without the top-$\psi$ vectors with the highest cosine distances, which are considered malicious. In this case, $\psi$ is either a known bound (i.e., defined in advance by the service provider) or an accepted percentile determined based on an assumed attacker ratio that follows a normal distribution.

Alg. 1 provides the formal details of our defense scheme, including support for quantized aggregation. Note that we use an optimizer with momentum for FedAvg which ensures that even if the majority of clients picked at random in a training round happens to be malicious, the optimization is still based on benign contributions from the previous round. App. B.3 provides details on the sub-protocols utilized in our defense algorithm given in Alg. 1.

5.1.2 Effectiveness

To analyze the effectiveness of ScionFL-Aura, we test it against the Min-Max attack [112].

**Setup.** Each training involves $N = 50$ clients while $-$ as in [112] $-$ 20% of them are corrupted.8 Per training iteration, a random subset of $n = 10$ clients is chosen to train the global model. Each client C runs its local training for 10 iterations with batches of $B = 128$ samples and a learning rate

$\mu_{th}$ is set to 3 and the momentum is 0.9.

**Experimental Results.** Our results when training ResNet9 on CIFAR10 (1) without an attack, (2) under attack without defense, and (3) under attack with ScionFL-Aura in place are given in Fig. 10. We compare the attack’s effect when no compression is in place as well as when applying SQ with the randomized Hadamard transform (HSQ) or with Kashin’s representation (KSQ). Next to the ResNet9 architecture, we also provide results for training VGG11 in App. §B.3.2. As shown in Fig. 10, our re-implementation of the Min-Max attack substantially reduces the validation accuracy by up to 20% when no defense is in place. This is in line with [112], where the authors report an accuracy degradation between 10.1% and 42.1% for CIFAR10, depending on the network architecture and the aggregation scheme. Furthermore, our experiments show that quantization does not significantly change the impact of the attack. When our defense ScionFL-Aura is enabled, we can remove more than half of the malicious updates in each training iteration compared to when no defense is in place. In fact, quantization supports our defense as the additional noise added to synchronized malicious updates overturns the attackers’ ability of staying just below the detection threshold. As a result, compared to unprotected training, the validation accuracy decreases by at most 7.7% for HSQ and 10.7% for KSQ.

---

8 Scaling a quantized vector requires simply scaling the scales (cf.§2.4).

9 [113] points out that assuming more than 1% of corrupted clients is unrealistic for most scenarios. However, in our experiments the attack failed to notably reduce the accuracy with such a low corruption level. Thus, we tested against 20% of corrupted clients as in the original attack paper [112].
Figure 10: Effect of Min-Max attack [112] on training ResNet9 with CIFAR10 with and without our defense ScionFL-Aura assuming 20% of \( N = 50 \) clients are corrupted. The number of attackers included in the global update varies even without defense due to random client selection.

### 5.1.3 MPC-friendly Variant

A naive secure realization of ScionFL-Aura outlined in Alg. 1 utilizing MPC will yield an inefficient solution, particularly over a ring architecture. This is due to some of the algorithm’s non-MPC friendly primitives, which are discussed below.

1. **(Line 5 in Alg. 1).** The computation of \( L_2 \)-norm within \( L_2\text{-NORM} \) (cf. Alg. 3 in App. B.3) involves calculating the square root of a ring element, which corresponds to a decimal value. To alleviate this, we ask the clients to submit the \( L_2 \)-norm of their gradient vectors and the MPC servers verify them. To be more specific, the provided \( L_2 \)-norm is squared and compared to a squared-\( L_2 \)-norm computed by the MPC servers via a secure comparison protocol [30, 86].

2. **(Lines 11 & 12 in Alg. 1).** When using \( L_2 \)-norm scaling, the scales of the gradient vector must be bounded if the corresponding \( L_2 \)-norm is greater than the limit. In particular, the procedure entails dividing the vector by its \( L_2 \)-norm. Because division is expensive in MPC over rings, we ask the client to submit the reciprocal of the \( L_2 \)-norm as well, similar to the method suggested above. The provided value is validated by multiplying it by the \( L_2 \)-norm supplied by the client and checking whether the product is a 1.

3. **(Line 16 in Alg. 1).** The calculation of the cosine-distance between the gradient vector and the reference \( \vec{X}^{agg} \) requires computing the \( L_2 \)-norm of \( \vec{X}^{agg} \) and dividing by it, as shown in \textsc{Cosine} in Alg. 4. However, cosine distances are used to filter out the top-\( \psi \) vectors with the highest cosine distance, as shown in Alg. 1 (Line 18). As a result, we may safely disregard the division by the \( L_2 \)-norm of \( \vec{X}^{agg} \) when computing the cosine distance for our purpose.

In addition to the aforementioned optimizations, we notice that most of the values computed as part of the \( \vec{X}^{agg} \) computation in the \textsc{Aggregate} function (Line 2 in Alg. 1) can be reused in the next steps, thus lowering the overhead of the defense scheme over simple aggregation.

### 6 Conclusion

We propose ScionFL, the first protocol for secure aggregation that supports any linear quantization scheme. Furthermore, we designed the first approximate MPC protocol for bit-to-arithmetic conversions, which might be of interest for a wide range of other applications that can tolerate or even desire noise in their computation. Additionally, we designed a novel defense named ScionFL-Aura against state-of-the-art untargeted model poisoning attacks and outlined its efficient implementation in MPC. As part of future work, we plan to implement and evaluate ScionFL as well as ScionFL-Aura in an established MPC framework.

### Acknowledgments

This project received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/ 251805230, by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within ATHENE, and by the Private AI Collaborative Research Institute, funded by Intel, Avast, and VMware.
References

[1] J. Aas and T. Geoghegan. Introducing ISRG Prio Services for Privacy Respecting Metrics. https://www.abetterinternet.org/post/introducing-prio-services/. 2

[2] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou. Prio+: Privacy Preserving Aggregate Statistics via Boolean Shares. In SCN, 2022. 2, 5, 6, 8, 19, 23, 24, 25

[3] A. F. Aji and K. Heafield. Sparse communication for distributed gradient descent. In EMNLP, 2017. 2, 19

[4] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli. The Convergence of Sparsified Gradient Methods. In NeurIPS, 2018. 1, 2, 19

[5] S. Andreina, G. A. Marson, H. Möllering, and G. Karame. BaFFLe: Backdoor Detection via Feedback-based Federated Learning. In IEEE ICDCS, 2021. 21

[6] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In ACM CCS, 2016. 4, 5

[7] G. Asharov, S. Halevi, Y. Lindell, and T. Rabin. Privacy-Preserving Search of Similar Patients in Genomic Data. PETS, 2018, 2018. 19

[8] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In ACM CCS, 2013. 19, 22

[9] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How To Backdoor Federated Learning. In AIJSTATS, 2020. 3, 11, 12, 20

[10] G. Baruch, M. Baruch, and Y. Goldberg. A Little Is Enough: Circumventing Defenses For Distributed Learning. In NeurIPS, 2019. 3, 11, 20

[11] R. B. Basat, M. Mitzenmacher, and S. Vargaftik. How to Send a Real Number Using a Single Bit (And Some Shared Randomness). In ICALP, 2021. 1, 5

[12] R. B. Basat, S. Vargaftik, A. Portnoy, G. Einziger, Y. Ben-Itzhak, and M. Mitzenmacher. QUIC-FL: Quick Unbiased Compression for Federated Learning. 2022. https://arxiv.org/abs/2205.13341. 4

[13] C. Baum, I. Damgård, T. Toft, and R. W. Zakarias. Better Preprocessing for Secure Multiparty Computation. In ACNS, 2016. 5

[14] C. Beguier, M. Andreux, and E. W. Tramel. Efficient Sparse Secure Aggregation for Federated Learning. 2020. https://arxiv.org/abs/2007.14861. 2, 4, 6, 9, 10, 20, 24, 25

[15] 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. 2, 19, 20

[16] A. Ben-Efraim, M. Nielsen, and E. Omri. Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing. In ACNS, 2019. 8, 21

[17] J. Bernstein, Y. Wang, K. Azizzadenesheli, and A. Anandkumar. SIGNSGD: Compressed Optimisation for Non-Convex Problems. In ICML, 2018. 1, 2, 3, 9, 20

[18] A. Beznosikov, S. Horváth, P. Richtárik, and M. Safaryan. On Biased Compression for Distributed Learning. 2020. https://arxiv.org/abs/2002.12410. 2, 19

[19] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. B. Calo. Analyzing Federated Learning through an Adversarial Lens. In ICML, 2019. 11, 20

[20] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In NeurIPS, 2017. 20, 21

[21] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame. MP2ML: a mixed-protocol machine learning framework for private inference. In ARES, 2020. 4

[22] F. Boenisch, A. Dziedzic, R. Schuster, A. S. Shamsabadi, I. Shumailov, and N. Papernot. When the Curious Abandon Honesty: Federated Learning Is Not Private. 2021. https://arxiv.org/abs/2112.02918. 2, 20

[23] K. A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In ACM CCS, 2017. 1, 2, 20

[24] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. In CRYPTO, 2019. 19, 22

[25] B. Bozdemir, S. Canard, O. Ermis, H. Möllering, M. Önen, and T. Schneider. Privacy-preserving
Density-based Clustering. In ACM ASIA CCS, 2021.

[26] L. Braun, D. Demmler, T. Schneider, and O. Tkachenko. MOTION - A Framework for Mixed-Protocol Multi-Party Computation. ACM Trans. Priv. Secur., 2022. 19

[27] M. Byali, H. Chaudhari, A. Patra, and A. Suresh. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. PETS, 2020. 19

[28] S. Caldas, J. Konečný, H. B. McMahan, and A. Talwalkar. Expanding the Reach of Federated Learning by Reducing Client Resource Requirements. 2018. http://arxiv.org/abs/1812.07210. 1, 2, 4, 24

[29] X. Cao, M. Fang, J. Liu, and N. Z. Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. In NDSS, 2021. 12, 21

[30] O. Catrina and A. Saxena. Secure Computation with Fixed-Point Numbers. In FC, 2010. 13, 19, 22

[31] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh. ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction. In ACM CCSW@CCS, 2019. 5, 19

[32] H. Chaudhari, R. Rachuri, and A. Suresh. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In NDSS, 2020. 8, 19

[33] W. Chen, C. A. Choquette-Choo, P. Kairouz, and A. T. Suresh. The Fundamental Price of Secure Aggregation in Differentially Private Federated Learning. In ICML, 2022. 1, 2, 20

[34] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. In ASIACRYPT. Springer, 2017. 19

[35] B. Chmiel, L. Ben-Uri, M. Shkolnik, E. Hoffer, R. Banner, and D. Soudry. Neural gradients are near-lognormal: improved quantized and sparse training. In ICLR, 2021. 7

[36] H. Corrigan-Gibbs and D. Boneh. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In USENIX NSDI, 2017. 2, 5, 19

[37] G. Couteau, P. Rindal, and S. Raghubaraman. Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes. In CRYPTO, 2021. 22, 24

[38] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. SPDZ$k$: Efficient MPC mod $2^k$ for Dishonest Majority. In CRYPTO, 2018. 19

[39] I. Damgård, D. Escudero, T. K. Frederiksen, M. Keller, P. Scholl, and N. Volgushev. New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning. In IEEE S&P, 2019. 19

[40] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. In ESORICS, 2013. 5

[41] I. Damgård, C. Orlandi, and M. Simkin. Yet Another Compiler for Active Security or: Efficient MPC Over Arbitrary Rings. In CRYPTO, 2018. 5

[42] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In CRYPTO, 2012. 2, 5

[43] P. Davies, V. Gurunanthan, N. Moshrefi, S. Ashkboos, and D. Alistarh. New bounds for distributed mean estimation and variance reduction. In ICLR, 2021. 4

[44] D. Demmler, T. Schneider, and M. Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In NDSS, 2015. 19

[45] Y. Dong, X. Chen, K. Li, D. Wang, and S. Zeng. FLOD: oblivious defender for private byzantine-robust federated learning with dishonest-majority. In ESORICS, 2021. 4, 21

[46] T. Elahi, G. Danezis, and I. Goldberg. PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks. In ACM CCS, 2014. 19

[47] Z. Erkin, J. R. Troncoso-Pastoriza, R. L. Lagendijk, and F. Pérez-González. Privacy-Preserving Data Aggregation in Smart Metering Systems: An Overview. IEEE Signal Process. Mag., 2013. 19

[48] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl. Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits. In CRYPTO, 2020. 19

[49] M. Fang, X. Cao, J. Jia, and N. Z. Gong. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In USENIX Security, 2020. 3, 11, 20

[50] J. Fei, C. Ho, A. N. Sahu, M. Canini, and A. Sapio. Efficient sparse collective communication and its application to accelerate distributed deep learning. In ACM SIGCOMM Conference, 2021. 2, 19

[51] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllerling, T. D. Nguyen, P. Rieger, A. Sadeghi, T. Schneider, H. Yalame, and S. Zeitouni. SAFELearn: Secure Aggregation for private FEderated
[52] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein. High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority. In *EUROCRYPT*, 2017. 5

[53] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov. Property Inference Attacks on Fully Connected Neural Networks using Permutation Invariant Representations. In *ACM CCS*, 2018. 1

[54] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *ACM STOC*, 1987. 19

[55] S. D. Gordon, S. Ranellucci, and X. Wang. Secure Computation with Low Communication from Cross-Checking. In *ASIACRYPT*, 2018. 8

[56] V. Goyal, H. Li, R. Ostrovsky, A. Polychroniadou, and Y. Song. ATLAS: Efficient and Scalable MPC in the Honest Majority Setting. In *CRYPTO*, 2021. 19

[57] A. Hegde, H. Möllering, T. Schneider, and H. Yalam. SoK: Efficient Privacy-preserving Clustering. In *PETS*, 2021. 4

[58] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright. Privacy-preserving Machine Learning as a Service. In *PETS*, 2018. 19

[59] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung. On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality. In *IEEE EuroS&P*, 2020. 2

[60] N. Ivkin, D. Rothchild, E. Ullah, V. Braverman, I. Stoica, and R. Arora. Communication-efficient Distributed SGD with Sketching. In *NeurIPS*, 2019. 19

[61] R. Jin, Y. Huang, X. He, H. Dai, and T. Wu. Stochastic Sign SGD for Federated Learning with Theoretical Guarantees. 2020. https://arxiv.org/abs/2002.10940. 3

[62] M. Joye and B. Libert. A Scalable Scheme for Privacy-Preserving Aggregation of Time-Series Data. In *FC*, 2013. 19

[63] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security*, 2018. 4

[64] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramchandran. FastSecAgg: Scalable Secure Aggregation for Privacy-Preserving Federated Learning. 2020. https://arxiv.org/abs/2009.11248. 1, 20

[65] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, and et al. Advances and Open Problems in Federated Learning. *Found. Trends Mach. Learn.*, 2021. 1, 2, 19

[66] S. Kamara and M. Raykova. Secure outsourced computation in a multi-tenant cloud. In *IBM Workshop on Cryptography and Security in Clouds*, 2011. 19

[67] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *ICML*, 2020. 3

[68] M. Keller, E. Orsini, and P. Scholl. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *ACM SIGSAC*, 2016. 19, 22

[69] M. Keller, V. Pastro, and D. Rotaru. Overdrive: Making SPDZ Great Again. In *EUROCRYPT*, 2018. 19

[70] M. Keller, P. Scholl, and N. P. Smart. An architecture for practical actively secure MPC with dishonest majority. In *ACM CCS*, 2013. 5, 19

[71] D. Kim, Y. Son, D. Kim, A. Kim, S. Hong, and J. H. Cheon. Privacy-preserving Approximate GWAS computation based on Homomorphic Encryption. 2019. https://eprint.iacr.org/2019/152. 19

[72] V. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *ICALP*, 2008. 19

[73] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated Learning: Strategies for Improving Communication Efficiency. 2016. http://arxiv.org/abs/1610.05492. 2, 3, 19

[74] N. Koti, M. Pancholi, A. Patra, and A. Suresh. SWIFT: Super-fast and robust privacy-preserving machine learning. In *USENIX Security*, 2021. 9

[75] N. Koti, S. Patil, A. Patra, and A. Suresh. MPClan: Protocol suite for privacy-conscious computations, 2022. https://eprint.iacr.org/2022/675. 5, 8

[76] N. Koti, A. Patra, R. Rachuri, and A. Suresh. Tetrad: Actively Secure 4PC for Secure Training and Inference. In *NDSS*, 2022. 9, 19

[77] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 10

[78] K. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *ICALP*, 2008. 19

[79] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated Learning: Strategies for Improving Communication Efficiency. 2016. http://arxiv.org/abs/1610.05492. 2, 3, 19

[80] N. Koti, M. Pancholi, A. Patra, and A. Suresh. SWIFT: Super-fast and robust privacy-preserving machine learning. In *USENIX Security*, 2021. 9

[81] N. Koti, S. Patil, A. Patra, and A. Suresh. MPClan: Protocol suite for privacy-conscious computations, 2022. https://eprint.iacr.org/2022/675. 5, 8

[82] N. Koti, A. Patra, R. Rachuri, and A. Suresh. Tetrad: Actively Secure 4PC for Secure Training and Inference. In *NDSS*, 2022. 9, 19

[83] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 10

[84] K. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *ICALP*, 2008. 19

[85] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated Learning: Strategies for Improving Communication Efficiency. 2016. http://arxiv.org/abs/1610.05492. 2, 3, 19

[86] N. Koti, M. Pancholi, A. Patra, and A. Suresh. SWIFT: Super-fast and robust privacy-preserving machine learning. In *USENIX Security*, 2021. 9

[87] N. Koti, S. Patil, A. Patra, and A. Suresh. MPClan: Protocol suite for privacy-conscious computations, 2022. https://eprint.iacr.org/2022/675. 5, 8

[88] N. Koti, A. Patra, R. Rachuri, and A. Suresh. Tetrad: Actively Secure 4PC for Secure Training and Inference. In *NDSS*, 2022. 9, 19
[79] M. Lam, G. Wei, D. Brooks, V. J. Reddi, and M. Mitzenmacher. Gradient disaggregation: Breaking privacy in federated learning by reconstructing the user participant matrix. In ICML, 2021. 2, 20

[80] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proc. IEEE, 1998. 2, 10, 24

[81] K. H. Li, P. P. B. de Gusmão, D. J. Beutel, and N. D. Lane. Secure aggregation for federated learning in flower. In ACM International Workshop on Distributed Machine Learning, 2021. 1

[82] L. Li, W. Xu, T. Chen, G. B. Giannakis, and Q. Ling. RSA: Byzantine-Robust Stochastic Aggregation Methods for Distributed Learning from Heterogeneous Datasets. In AAAI, 2019. 21

[83] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated Optimization in Heterogeneous Networks. In MLSys, 2020. 3

[84] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai. Efficient Constant Round Multi-party Computation Combining BMR and SPDZ. In CRYPTO, 2015. 21

[85] Y. Lyubarskii and R. Vershynin. Uncertainty principles and vector quantization. IEEE Trans. Inf. Theory, 2010. 4

[86] E. Makri, D. Rotaru, F. Vercauteren, and S. Wagh. Rabbit: Efficient Comparison for Secure Multi-Party Computation. In FC, 2021. 13

[87] M. Mansouri, M. Önen, W. Ben Jaballah, and M. Conti. SoK: Secure aggregation based on cryptographic schemes for federated Learning. In PETS, 2023. 3, 20

[88] S. Mazloom, P. H. Le, S. Ranellucci, and S. D. Gordon. Secure parallel computation on national scale volumes of data. In USENIX Security, 2020. 8

[89] 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. 1, 3, 8

[90] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. In IEEE S&P, 2019. 1

[91] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa. Delphi: A cryptographic inference service for neural networks. In USENIX Security, 2020. 4

[92] P. Mohassel and P. Rindal. ABY³: A Mixed Protocol Framework for Machine Learning. In ACM CCS, 2018. 5, 8, 9, 22

[93] P. Mohassel and Y. Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In IEEE S&P, 2017. 19, 22

[94] A. Mondal, Y. More, P. Ramachandran, P. Panda, H. Virk, and D. Gupta. Scotch: An Efficient Secure Computation Framework for Secure Aggregation. 2022. https://arxiv.org/abs/2201.07730. 1, 2, 4

[95] J. Münch, T. Schneider, and H. Yalame. VASA: Vector AES Instructions for Security Applications. In ACM ACSAC, 2021. 19

[96] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In IEEE S&P, 2019. 2, 20

[97] T. D. Nguyen, P. Rieger, H. Chen, H. Yalame, H. Möller, H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, F. Koushanfar, A.-R. Sadeghi, T. Schneider, and S. Zeitouni. FLAME: Taming Backdoors in Federated Learning. In USENIX Security, 2022. 2, 4, 11, 12, 21, 26

[98] A. E. Ouadrhiri and A. Abdelhadi. Differential Privacy for Deep and Federated Learning: A Survey. IEEE Access, 2022. 1

[99] A. Patra, P. Sarkar, and A. Suresh. Fast Actively Secure OT Extension for Short Secrets. In NDSS, 2017. 22

[100] A. Patra, T. Schneider, A. Suresh, and H. Yalame. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In USENIX Security, 2021. 4, 19, 22

[101] A. Patra, T. Schneider, A. Suresh, and H. Yalame. Syncirc: Efficient Synthesis of Depth-Optimized Circuits for Secure Computation. In IEEE HOST, 2021. 19

[102] A. Patra and A. Suresh. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In NDSS, 2020. 8, 9

[103] R. A. Popa, A. J. Blumberg, H. Balakrishnan, and F. H. Li. Privacy and accountability for location-based aggregate statistics. In ACM CCS, 2011. 19

[104] A. Pyrgelis, C. Troncoso, and E. D. Cristofaro. Knock Knock, Who’s There? Membership Inference on Aggregate Location Data. In NDSS, 2018. 1

[105] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar. XONN: XNOR-based Oblivious Deep Neural Network Inference. In USENIX Security, 2019. 19
[106] P. Richtárik, I. Sokolov, and I. Fatkhullin. EF21: A New, Simpler, Theoretically Better, and Practically Faster Error Feedback. In NeurIPS, 2021. 2, 19

[107] D. Rotaru and T. Wood. MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security. In INDOCRYPT, 2019. 22, 23

[108] 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. 1, 2, 20

[109] M. Safaryan, E. Shulgin, and P. Richtárik. Uncertainty Principle for Communication Compression in Distributed and Federated Learning and the Search for an Optimal Compressor. 2020. https://arxiv.org/abs/2002.08958. 4

[110] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux. POSEIDON: Privacy-preserving federated neural network learning. In NDSS, 2021. 20

[111] 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. 1, 2, 3, 19

[112] V. Shejwalkar and A. Houmansadr. Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning. In NDSS, 2021. 2, 3, 11, 12, 13, 20, 21, 26

[113] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage. Back to the Drawing Board: A Critical Evaluation of Poisoning Attacks on Production Federated Learning. In IEEE S&P, 2022. 11, 12, 20

[114] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership Inference Attacks Against Machine Learning Models. In IEEE S&P, 2017. 1

[115] J. So, B. Güler, and A. S. Avestimehr. TurboAggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning. IEEE J. Sel. Areas Inf. Theory, 2021. 20

[116] S. U. Stich, J.-B. Cordonnier, and M. Jaggi. Sparsified SGD with Memory. In NeurIPS, 2018. 2, 19, 20

[117] I. Stoica and S. Shenker. From cloud computing to sky computing. In ACM HotOS ’21: Workshop on Hot Topics in Operating Systems, 2021. 2

[118] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan. Can you really backdoor federated learning? In NeurIPS FL Workshop, 2019. 12, 20

[119] A. Suresh. MPCLeague: Robust MPC Platform for Privacy-Preserving Machine Learning. PhD Thesis, 2021. https://arxiv.org/abs/2112.13338. 8, 9, 21

[120] A. T. Suresh, F. X. Yu, S. Kumar, and H. B. McMahan. Distributed Mean Estimation with Limited Communication. In ICML, 2017. 1, 2, 3, 4, 5, 19

[121] H. Tang, S. Gan, A. A. Awan, S. Rajbhandari, C. Li, X. Lian, J. Liu, C. Zhang, and Y. He. 1-bit Adam: Communication Efficient Large-Scale Training with Adam’s Convergence Speed. In ICML, 2021. 3

[122] S. Vargaftik, R. B. Basat, A. Portnoy, G. Mendelson, Y. Ben-Itzhak, and M. Mitzenmacher. EDEN: Communication-Efficient and Robust Distributed Mean Estimation for Federated Learning. In ICML, 2022. 2, 4, 19

[123] S. Vargaftik, R. Ben-Basat, A. Portnoy, G. Mendelson, Y. Ben-Itzhak, and M. Mitzenmacher. DRIVE: One-bit Distributed Mean Estimation. In NeurIPS, 2021. 1, 3, 4, 21

[124] J. Wang, Z. Charles, Z. Xu, G. Joshi, H. B. McMahan, and et al. A Field Guide to Federated Optimization. 2021. https://arxiv.org/abs/2107.06917. 2, 19

[125] L. Wang, S. Xu, X. Wang, and Q. Zhu. Eavesdrop the Composition Proportion of Training Labels in Federated Learning. 2019. http://arxiv.org/abs/1910.06044. 1

[126] 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. 4

[127] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is Feature Selection Secure against Training Data Poisoning? In ICML, 2015. 11

[128] C. Yang, J. So, C. He, S. Li, Q. Yu, and S. Avestimehr. LightSecAgg: Rethinking Secure Aggregation in Federated Learning. 2021. https://arxiv.org/abs/2109.14236. 1

[129] A. C.-C. Yao. Protocols for Secure Computations (Extended Abstract). In FOCS, 1982. 19

[130] D. Yin, Y. Chen, K. Ramchandran, and P. L. Bartlett. Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. In ICML, 2018. 20, 21

[131] Z. Zhang, A. Panda, L. Song, Y. Yang, M. W. Mahoney, P. Mittal, K. Ramchandran, and J. Gonzalez. Neurotoxin: Durable Backdoors in Federated Learning. In ICML, 2022. 3
A Related Work & Background Information

In this section, we review additional related work in the area of compression techniques, (approximate) MPC, secure aggregation, and FL attacks as well as defenses.

A.1 Additional Compression Techniques

In this work, we focus on quantization as a means to reduce bandwidth. We nevertheless briefly overview some additional techniques considered for FL gradient compression.

Sparsification. Some works consider sparsifying the gradients (e.g., [3, 50, 73, 116]). Quantization is mostly orthogonal to such techniques as it can be applied to the sparsified gradients. Namely, sparsification is designed to reduce the number of entries, whereas quantization reduces the number of bits used to represent each entry.

Client-side Memory-based Techniques. Some compression techniques, including Top-k [116] and sketching [60], rely on client-side memory and error-feedback [4, 18, 106, 111] to ensure convergence. We consider the cross-device FL setup where clients are stateless (e.g., a client may appear only once during a training procedure). Therefore, client-side-memory-based techniques are mostly designed for the cross-silo FL setup and are less applicable to cross-device FL.

Entropy Encodings. Some techniques use entropy encoding such as arithmetic encoding and Huffman encoding (e.g., [4, 120, 122]). While such techniques are appealing in their bandwidth-to-accuracy trade-offs, it is unclear how to allow for an efficient secure aggregation with such techniques as gradients must be decoded before being averaged. Also, such techniques usually incur a higher computational overhead at the clients than fixed-length representations.

Finally, an additional review of current state-of-the-art gradient compression techniques and some open challenges can be found in [65, 73, 124].

A.2 Secure Multi-party Computation

The field of secure multi-party computation (MPC) started with the seminal work of Yao [129] in 1982. It enables to securely compute arbitrary functions on private inputs without leaking anything beyond what can be inferred from the output. Since then, the field of MPC has seen a variety of advancements of used primitives effectively improving communication and computation efficiency, e.g., [8, 24, 48, 72, 95]. Also, tailored efficient optimizations for varying number of computation parties have been explored, e.g., [31, 32, 44, 76, 100, 101]. Moreover, MPC research considers different assumptions regarding adversarial behavior such as the well-known semi-honest [26, 44] and malicious security model [39, 68–70]) as well as numbers of corrupted computation parties (e.g., honest majority [54, 56] or dishonest majority/full threshold security [26, 38, 44, 70, 100]). Beyond running the computation among several non-colluding parties, another well-established system model (which we use in our work) is outsourcing, where the data owners secret-share their private input data among a set of non-colluding computing parties which then run the private computation on their behalf. In this scenario, the a semi-honest MPC protocol also protects against malicious behavior of data owners [66].

A.3 Approximate Secure Computation

To improve efficiency of MPC, few works already considered approximations of the exact computation. Such approximations in MPC include using integer or fixed-point instead of floating-point operations (too many works to cite), approximations in genomic computation [7], and in privacy-preserving machine learning such as for division [30], activation functions [27, 58, 93], and completely changing the classifier to be MPC-friendly [105]. Also, for FHE, approximations are used such as in the approximate HE scheme CKKS [34], which is implemented in the HEAN library and was used for approximate genomic computations in [71]. In this work, we propose for the first time to use approximations to substantially improve efficiency of FL when combined with MPC and give detailed evaluations on the errors introduced thereby.

A.4 Secure Aggregation & Inference Attacks

Performing secure aggregation without revealing anything about the aggregated input values beyond what can be inferred from the output was already investigated more than 10 years ago, for example, in the context of smart metering, e.g., [47, 78]. It has come a long way since then, resulting in practical solutions for real-world applications nowadays.

For example, Prio [36] introduces secure protocols for aggregate statistics such as sum, mean, variance, standard deviation, min/max, and frequency. It uses additive arithmetic secret sharing, offers full-threshold security among a small set of servers running the secure computation, and validates inputs to protect against malicious clients. Prio+ [2] optimizes client computation and communication compared to [36] with a Boolean secret sharing-based client input validation and an additional conversion from Boolean to arithmetic sharing. In our work, we optimize the naive bit-to-arithmetic conversion presented in [2] for our $f_{secAgg}$ protocol (cf. §4.1).

Bell et al. [15] propose the first secure aggregation protocols in both the semi-honest and malicious security model (tolerating server and client corruptions) with poly-logarithmic client communication and computation. Popa et al. [103] specifically focus on secure location-based aggregation statistics. Joye et al. [62] on time-series data, and PrivEx [46] on traffic data in anonymous communication.

Inference attacks. In FL, clients share locally trained model updates with a central party to contribute to the training of a global model. However, sharing those local models makes the system vulnerable to data leakage. Even a semi-honest
A common countermeasure against inference attacks is to use secure aggregation. As FL poses specific challenges such as a large number of clients and drop-out tolerance, tailored secure aggregation protocols for FL have been proposed [15, 23, 51, 64, 110, 115]. Those hide individual updates, ensuring that the server has only access to global updates, hence, effectively prohibiting the analysis of individual updates for inference attacks. The first scheme, SegAgg [23], combines secret sharing with authenticated symmetric encryption, but requires 4 communication rounds per training iteration among servers and client. Similar issues can also be observed for other existing secure aggregation protocols for FL as they incur significant computation and communication overhead: Fereidonni et al. [51] compare several secure aggregation protocols with respect to efficiency and practicality. Mansouri et al. [87] give a SoK of secure aggregation schemes w.r.t. their suitability for FL.

To the best of our knowledge, only Chen et al. [33] and Beguier et al. [14] have considered both compression and secure aggregation in combination for FL. Specifically, Chen et al. [33] combine SegAgg [23] with sparse random projection and count-sketching [108] for compression. Moreover, they add noise using a distributed discrete Gaussian mechanism to generate a differential private output. Beguier et al. [14] combine arithmetic secret-sharing with Top-k sparsiﬁcation [116] and 1-bit quantization [17]. As pointed out in §A.1, both sketching and sparsiﬁcation are suboptimal for our envisioned cross-device setting given that they require memory and error-feedback on the client side. In contrast, we focus on a dynamic scenario where clients might drop-out at any time and may contribute only once to the training. [14] offers more eﬁcient secure aggregation than using SegAgg [23], but we further improve client-side communication thanks to our (approximated) secure bit aggregation (cf. §3).

A.5 Poisoning Attacks & Defenses

Poisoning attacks can be categorized into untargeted and targeted attacks based on the goals of the attacker [49]. In the former case, the attacker aims to corrupt the global model so that it reduces or even destroys the performance of the trained model for a large number of test inputs, yielding a ﬁnal global model with a high error rate [10, 49, 112]. In the latter case, the attacker aims to activate attacker-deﬁned triggers that cause a victim model to do targeted misclassiﬁcations, which can then be activated in the inference phase [19, 118]. Notably, other classiﬁcation results without the trigger behave normally and main task accuracy remains high. The second class of attacks is sometimes also referred to as backdoor attacks [9]. As discussed in §5, we consider only untargeted poisoning following the argument in [113]: This class of attacks is particularly challenging as service providers may not notice they are under attack given they do not know which accuracy is achievable in a fresh training of a new model. Also, even small accuracy reductions can lead to serious economical losses.

Below, we detail three state-of-the-art untargeted poisoning attacks, LIE [10], Fang [49], and Shejwalkar et al. [112], which are most relevant to our work.

Little is Enough (LIE) attack [10]. In LIE [10], malicious clients manipulate their local updates by adding noise drawn from the normal distribution to the gradients of “clean” updates they created following the normal training process to cause a disorientation. LIE is assuming independent and identically distributed (iid) data and was tested against various robust aggregations such as trimmed-mean [130].

Fang et al. [49]. The authors of [49] formulate their untargeted poisoning attack as an optimization problem where the manipulated updates aim at maximally disorienting the global model from the benign direction. However, they assume the adversary to either know or guess the deployed (robust) aggregation mechanism. Additionally, the attack was shown to be ineffective for iid as well as severely unbalanced non-iid training datasets [112].

Shejwalkar and Houmansadr [112]. The attacks of Shejwalkar and Houmansadr [112] follow a similar idea as [49]: they maximize the distances between benign and malicious updates while using the evasion of outlier-based detection mechanisms as a boundary. Concretely, they formalize the “Min-Max” optimization problem as follows:

$$\text{argmax}_{\gamma} \max_{i \in [n]} \|\nabla^m_i - \nabla_i\|_2 \leq \max_{i,j \in [n]} \|\nabla_i - \nabla_j\|_2$$

$$\nabla^m = p_{avg}(\nabla_{\{i \in [n]\}}) + \gamma \nabla^p,$$
number of clients and $m$ the number of anticipated malicious clients. Multi-krum [20] extends this idea to a selection of $c$ (instead of just one) updates. Median [130] is another coordinate-wise aggregation selecting the coordinate-wise median of each update parameter. A straightforward idea to assess (to some extent) if a specific gradient is malicious is to use an auxiliary dataset (rootset) at the aggregator to validate the performance of the updated global model [29, 45, 82]. FLTrust [29] and FLOD [45] use the ReLU-clipped cosine-similarity/Hamming distance between each received update and the aggregator-computed baseline update based on the auxiliary dataset. RSA [82] uses an $\ell_1$-norm-based regularization, which is also comparing to the aggregator-computed baseline update. The recently proposed Divider and Conquer (DnC) aggregation [112] combines dimensionality reduction using random sampling with an outlier-based filtering.

The so far discussed poisoning defenses are not compatible with secure aggregation protocols in a straightforward manner or lead to an intolerable overhead. Only two works, namely FLAME [97] and BaFFLe [5] simultaneously consider both threats. Concretely, FLAME [97] uses a density-based clustering to remove updates with significantly different cosine distances (i.e., different directions) combined with clipping (for more subtle manipulations). \[10\] BaFFLe [5] introduces a feedback loop enabling a subset of clients to evaluate each global model update, while being compatible with arbitrary secure aggregation schemes.

In our work, we combine (1) communication efficiency due to quantization, (2) data privacy due to secure aggregation, and (3) robustness due to a novel poisoning defense.

## B Additional Information

This section provides additional details for our constructions. We begin with some preliminary information and details about the Hadamard and Kashin quantization techniques. This will be followed by information about the underlying MPC scheme used in our work, and finally, additional information about our framework, including benchmarks.

### B.1 Preliminaries

**Lemma B.1** (Expected Values). Given $n, p \in \mathbb{Z}$, we have

1. $\sum_{p=0}^{n} p \cdot \binom{n}{p} = n \cdot 2^{n-1}$.

2. $\sum_{p=0}^{\lfloor n/2 \rfloor} 2p \cdot \binom{n}{2p} = \sum_{p=0}^{\lfloor n/2 \rfloor} (2p+1) \cdot \binom{n}{2p+1} = n \cdot 2^{n-2}$.

**Proof.** Consider the binomial formula for $(1+y)^n$, given by

\[ \sum_{p=0}^{n} \binom{n}{p} y^p = (1+y)^n \quad (17) \]

Differentiating Eq. (17) with respect to $y$ will give

\[ \sum_{p=0}^{n} \binom{n}{p} p \cdot y^{p-1} = n \cdot (1+y)^{n-1} \quad (18) \]

Substituting $y = 1$ in Eq. (18) gives the first result (1). Similarly, setting $y = -1$ in Eq. (18) gives

\[ \sum_{p=0}^{n} (-1)^p \cdot \binom{n}{p} = 0 \quad (19) \]

Combining Eq. (19) with the first result (1) will give the second result (2).

### B.1.1 Overhead of HSQ and KSQ Quantization

For being able to use an efficient GPU-friendly implementation of the randomized Hadamard transform, which we use for both rotating the gradients in HSQ and for calculating Kashin’s coefficients in KSQ, we require that the gradients’ size to be a power of 2. A simple solution to meet this requirement is padding. For example, for the LeNet architecture with $\approx 60k$ parameters, we can pad the gradient to $2^{16} = 65536$ entries with a small resulting overhead of $\approx 6.2\%$ (i.e., using $\approx 1.06$ bits per coordinate instead of 1). However, a more sophisticated approach is to divide the gradient into decreasing power-of-two-sized chunks and inflate only the last (smallest) chunk.\[11\] For example, for the LeNet architecture, we can decompose it into chunks of size 32768, 16384, 8192, 4096, 512, that sum up to 61952 (with an additional overhead of two floats per chunk) with a resulting overhead of only $\approx 1.44\%$. Also, for Kashin’s representation, we use $\lambda = 1.15$ for each chunk (an extra 15\% of space) as used in previous works (e.g., [123]). To summarize, we state these resulting overheads in Tab. 3.

| Architecture  | $n$ | SQ   | HSQ  | KSQ  |
|---------------|-----|------|------|------|
| LeNet         | 61706 | 61706 | 62272 | 73024 |
| ResNet9       | 4903242 | 4903242 | 4915456 | 5767424 |
| ResNet18      | 11220132 | 11220132 | 11272192 | 12583040 |

Table 3: Exact number of bits used for different network architectures and quantization schemes compared to the baseline number of coordinates $n$.

### B.1.2 MPC Protocols

In this section, we go over the details of the underlying MPC protocols used in our scheme. We consider three MPC servers, $\mathcal{S} = \{S_1, S_2, S_3\}$, to which the clients delegate the aggregation computation, as shown in Fig. 1. All the operations are carried out in either an $\ell$-bit ring, $\mathbb{Z}_2^\ell$, or a binary ring, $\mathbb{Z}_2^*$. Before we go into the protocols, we provide additional details regarding the masked evaluation scheme [16, 84, 119] discussed in §4, starting with the sharing semantics.

\[\text{as FLAME contains some similar components as ScionFL-Aura, we give a detailed comparison in §B.3.3.}\]
Sharing Semantics. We use two different sharing schemes in our protocols:
1. \([-\cdot]-sharing.\) A value \(v \in \mathbb{Z}_{2^t}\) is said to be \([-\cdot]-shared\) among MPC servers in \(S\), if each server \(S_i\), for \(i \in [3]\), holds \(v_i \in \mathbb{Z}_{2^t}\) such that \(v_1 + v_2 + v_3 = v\).
2. \((\cdot)\)-sharing. In this sharing, every \(v \in \mathbb{Z}_{2^t}\) is associated with two values: a random mask \(\lambda_v \in \mathbb{Z}_{2^t}\) and a masked value \(m_v \in \mathbb{Z}_{2^t}\), such that \(v = m_v + \lambda_v\). Here, the share of an MPC server is defined as a tuple of the form \((m_v, [\lambda_v])\).

Handling Decimal Values. While the MPC protocol that we utilize is designed over a ring architecture, the underlying FL algorithms handle decimal numbers. To address this compatibility issue, we employ the well-known Fixed-Point Arithmetic (FPA) technique \([30, 92, 93]\), which encodes a decimal number in \(\ell\)-bits using the 2’s complement representation. The sign bit is represented by the most significant bit, while the \(f\) least significant bits are kept for the fractional component. We use \(\ell = 32\) bit values with \(f = 16\) in this work.

We will now go over the MPC protocols used in our scheme. The protocols are presented in a generic manner because our approach is not restricted to any specific MPC setting. Hence, some of the sub-protocols are treated as black-boxes that can be instantiated using any efficient protocols in the underlying MPC setting. Also, we assume that the protocols’ inputs are in \((\cdot)\)-shared form, and that the output is generated in \([-\cdot]\)-shared form among the MPC servers.

Inner Product Computation. Given two \(d\)-sized column vectors \(\mathbf{X}\) and \(\mathbf{Y}\), protocol \(\Pi_{Ip}\) computes the inner product of the two vectors, defined as \(z = \mathbf{X} \odot \mathbf{Y} = \sum_{i=1}^{d} x_i^i \cdot y_i^j\). For simplicity, consider the multiplication of two values \(x, y \in \mathbb{Z}_{2^t}\) as per the \((\cdot)\)-sharing semantics. We have
\[
z = xy = (m_x + \lambda_x)(m_y + \lambda_y) = m_x m_y + m_x \lambda_y + m_y \lambda_x + \lambda_x \lambda_y.
\]
Since the \(\lambda\) values are independent of the underlying secret, the servers can compute \([-\cdot]\)-shares of the term \(\lambda_x \lambda_y\) during preprocessing. This enables the servers to locally compute \([-\cdot]\)-shares of \(z\) during the online phase.

In addition to the above observation, since we operate over FPA representation, truncation \([30, 93]\) must be performed in order to keep the result \(z\) in FPA format after a multiplication. For this, we use the truncation pair method \([92]\), wherein a tuple of the form \((r, r/2^f)\) is generated in \((\cdot)\)-shared form among the servers during the preprocessing. Then, with very high probability, we have
\[
z/2^f = (z - r)/2^f + r/2^f.
\]

Hence, during the online phase, servers publicly open the value \((z - r)\) and apply the above transformation to obtain the \((\cdot)\)-shares of truncated \(z\), completing the protocol.

Now, in the case of the inner-product, the task can be split into \(d\) multiplications and the result obtained accordingly. Furthermore, because the desired result is the sum of the individual multiplication results, servers can sum them and communicate in a single shot, saving communication cost \([100]\).

The formal protocol appears in Fig. 11.

Bit-to-Arithmetic Protocol. Given the Boolean sharing of \(b \in \mathbb{Z}_2\), protocol \(\Pi_{BitA}\) computes the arithmetic sharing of the bit \(b\) over \(\mathbb{Z}_{2^t}\). As shown in Eq. 12, the arithmetic equivalent \(\tilde{b}\) for a bit \(b = m_b \oplus \lambda_b\) can be obtained as
\[
\tilde{b} = m_b \oplus \lambda_b = M_b + (1 - 2m_b) \cdot \lambda_b.
\]
Here, \(M_b\) and \(\lambda_b\) denote the arithmetic equivalents of \(m_b\) and \(\lambda_b\) respectively. In our protocol shown in Fig. 12, MPC servers invoke \(\Pi_{Pre}\) protocol on the Boolean \([-\cdot]\)-shares of \(\lambda_b\) in the preprocessing phase to obtain its respective arithmetic shares. This enables the servers to locally compute an additive shares of \(\tilde{b}\) during the online phase, as shown above. The rest of the steps proceed similar to the case with inner-product protocol and we omit the details.

To instantiate \(\Pi_{BitA}\), we use SPDZ-style computations \([68, 107]\), where oblivious transfer (OT) instances \([8, 24, 37, 99]\) are used among every pair of servers. Let \(\Pi_{OT}\) denote an instance of 1-out-of-2 OT with \(S_i\) being the sender and \(S_j\) being the receiver. Here, \(S_i\) inputs the sender messages \((x_0, x_1)\) while \(S_j\) inputs the receiver choice bit \(c \in \mathbb{Z}_2\) and obtains \(x_c\) as the output, for \(x_0, x_1 \in \mathbb{Z}_{2^t}\).

![Figure 11: Inner product protocol.](image)

![Figure 12: Bit-to-arithmetic conversion protocol.](image)
a 3-XOR using a daBit-style approach [107], but would result in 12 executions of 1-out-of-2 OTs. However, as pointed out in Prio+ [2], the cost could be further optimized due to the semi-honest security model being considered in this work rather than the malicious in [107]. Since Prio+ operates over two MPC servers, we extend their optimized daBit-generation protocol (cf. [2, daBitGen]) to our setting with three servers.

Given two bits \( b_1, b_2 \in \mathbb{Z}_2 \), the arithmetic share corresponding to their product can be generated using one instance of \( \Pi^\text{OT}_{\text{col}} \) with \( (x_0 = r, x_1 = r + b_1) \) as the OT-sender messages and \( b_2 \) as the OT-receiver choice bit. With this observation and using Eq. 7, servers can compute \( [\cdot] \)-shares corresponding to the bit \( \lambda_b \) using five OT invocations. The formal details appear in Fig. 13.

![Figure 13: Bit-to-arithmetic preprocessing.](image)

For the case of approximate bit conversion discussed in §3.1, the number of OT instances can be further reduced to three following Eq. 9. Concretely, the conversion involves computation of just \([b_1]_2 [b_2]_2 [b_3]_2\) and hence the OT instances II & III described in Fig. 13 are no longer needed.

When computing the sum of bits directly, the online communication can be optimized following inner-product protocol and the resulting protocol \( \Pi^\text{IP}_{\text{Bit}} \) is given in Fig. 14.

### Bit Injection Protocol

Given a Boolean vector \( \vec{M}_{d \times 1} \) and an arithmetic vector \( \vec{N}_{d \times 1} \) in the secret-shared form, protocol \( \Pi_{\text{Bit}} \) computes the inner product of the two vectors, defined as \( z = \vec{M} \odot \vec{N} \). This protocol is similar to the inner product protocol \( \Pi_{\text{IP}} \) presented in Fig. 11, with the main difference being that \( \vec{M} \) is a Boolean vector.

During the preprocessing phase, servers first generate the arithmetic shares of \( \lambda_M \) from its Boolean shares, similar to the bit-to-arithmetic protocol \( \Pi_{\text{BitA}} \) in Fig. 12. The remaining steps are similar to \( \Pi_{\text{IP}} \) in Fig. 11 and we omit the details.

![Figure 15: Bit injection (sum) protocol.](image)

### B.2 ScionFL: Additional Details

This section provides addition details of our FL framework ScionFL presented in §4.

#### B.2.1 Efficiency of Approximate Bit Conversion

In this section, we measure the efficiency gains achieved by our approximation method, discussed in §3.1, by counting the number of cross terms that must be computed securely using MPC. Cross terms are terms that compute the product of two or more shares. While the exact amount of computation and communication varies depending on the MPC protocol and setting (e.g., honest vs. dishonest majority or semi-honest vs. malicious security), we believe cross terms can provide a protocol-independent and realistic assessment of scalability.\(^{12}\)

Tab. 4 provides the details regarding the number of cross terms involved in obtaining the arithmetic equivalent of \( b = \bigoplus_{i=0}^{d-1} b_i \). The gains increase significantly with a higher number of shares \( q \) due to the exponential growth in the number of

---

\(^{12}\)We acknowledge that the analysis cannot provide an exact comparison, owing to the presence of the product term in the approximation. For example, depending on the underlying MPC setup, the product term (term\(_{\text{IP}}\)) may require more communication than the middle terms (term\(_{\text{m}}\)), and therefore the effect of approximation may be minimized.
Table 4: Efficiency analysis via approximate bit conversion with respect to the #cross-terms involved.

| Computation          | #cross-terms |
|----------------------|--------------|
| Exact (b)            | Approximate (b) |
| Bit-to-Arithmetic    | $2^n - q - 1$ | $q^2 - q + 1$ |
| Bit Injection        | $2^n + q^2 - 2q - 1$ | $q^2 - q + 1$ |

cross terms for the exact computation. Tab. 4 also provides the details for a bit injection operation in which the product of a Boolean bit $b$ and a scale value $s$ is securely computed. Given $s = \sum_{i=1}^n s_i$, the value $b \cdot s$ can be computed by first computing either $b$ or $b$ (depending on whether an exact or approximate value is required) and then multiplying by $s$.

B.2.2 Secure Bit Aggregation with Global Scales

Here, we consider the secure bit aggregation problem in the context of global scales, as discussed in (cf. §2.2). As shown in Eq. (6), the computation becomes simpler in the case of global scales since all clients utilize the same set of public scales, denoted by $s_{\text{min}}$ and $s_{\text{max}}$, to compute their quantized vector that corresponds to the rows of $\mathbf{B}$. Hence, we just need to compute the column-wise aggregate of the $\mathbf{B}$ matrix and use protocol $\Pi_{\text{BitA}}^B$ (Fig. 14 in §B.1.2) to do so. The resulting protocol $\Pi_{\text{Global SecAgg}}^B$ appears in Fig. 16.

![Figure 16: Secure aggregation – Global Scales.](image)

B.2.3 Secure Bit Aggregation with Local Scales

In §3.2, we provide an empirical accuracy evaluation for our approximate secure bit aggregation using global scales. Here, in Fig. 17, we additionally provide results considering local scales. In contrast to global scales, we can observe that for stochastic quantization without rotation the effect on the NMSE is reduced from three to one order of magnitude. Also, for rotation-based algorithms there are significant concrete improvements.

B.2.4 Detailed Communication Costs

In this section, we provide more detailed insights into the concrete communication costs for our secure aggregation protocols described in §4.1.

![Figure 17: NMSE comparison between exact and approximation-based aggregation for SQ, Hadamard SQ (HSQ), and Kashin SQ (KSQ) for local scales with $q = 4$ shares, various vector dimensions $d$, and number of clients $n$.](image)

| Approach          | Offline | Online |
|-------------------|---------|--------|
| Approach-I        | $n \cdot \text{BitA}^{\text{pre}} + n \cdot \text{Mult}^\pre$ | $n \cdot \text{BitA}^\text{on} + n \cdot \text{Mult}^\text{on}$ |
| Approach-II       | $n \cdot \text{BitA}^{\text{pre}} + n \cdot \text{Mult}^\pre$ | $n \cdot \text{Mult}^\text{on}$ |
| Approach-III      | $n \cdot \text{BitA}^\text{on} + \text{Mult}^\text{on}$ | $n \cdot \text{Mult}^\text{on}$ |

![Table 5: Communication costs aggregating quantized vectors with a single dimension for $n$ clients. Protocols $\Pi_{\text{BitA}}$ and $\Pi_{\text{Mult}}$ are treated as black-boxes, and their costs are represented as BitA and Mult, respectively. The superscript $\text{pre}$ in the costs denotes preprocessing and on denotes the online phase. We compare Approach-I (cf. Fig. 4 in §4.1.1), Approach-II (cf. Fig. 5 in §4.1.2), and Approach-III (cf. Fig. 6 in §4.1.3).](image)

First, in Tab. 5 we give a theoretical analysis for the communication cost when aggregating $n$ quantized vectors with a single dimension. Clearly, our Approach-III (cf. Fig. 6 in §4.1.3) is the most efficient, with the multiplication-related cost being completely independent of the number of clients $n$ due to the integration of SepAgg [14].

![Table 6: Inter-server communication per round in MiB for our MNIST/LeNet benchmark for different numbers of clients $n$ per round. Training is done using 1-bit SQ with Kashin’s representation (KQ). We compare Approach-I (cf. Fig. 4 in §4.1.1), Approach-II (cf. Fig. 5 in §4.1.2), and Approach-III (cf. Fig. 6 in §4.1.3). Additionally, we distinguish between using an exact bit-to-arithmetic conversion and our approximation (cf. §3.1).](image)

Next, in Tab. 6 we provide the detailed communication costs for the secure aggregation approaches discussed in §4.1 when training the LeNet architecture for image classification on the MNIST data set [80] using 1-bit SQ with Kashin’s representation [28]. We instantiate the OT instances required in the preprocessing phase, as discussed in §B.1.2, with silent OT [37], following Prio+ [2]. Here, we can observe...
the significant impact of including SepAgg [14] has in prac-
tice with performance improvements between Approach-II
and Approach-III of up to 16.6× in the offline phase.

| Approach           | n = 10^2 | n = 10^3 | n = 10^4 | n = 10^5 |
|--------------------|----------|----------|----------|----------|
| Prio+ [2]          | 9.45     | 94.50    | 945.04   | 9450.44  |
| Approach-III (Exact)| 3.94     | 39.42    | 394.17   | 3941.66  |
| Approach-III (Approx.) | 2.37     | 23.75    | 237.45   | 2374.53  |

Table 7: Total communication in MiB of our Approach-III (cf. Fig. 6
in §4.1.3) compared to Prio+ [2] to calculate the sum of bits for differ-
ent numbers of clients n and dimension m = 1000. For our Approach-
III, we distinguish between using an exact bit-to-arithmetic conver-
sion as in Prio+ [2] and our approximation (cf. §3.1).

Finally, in Tab. 7, we compare the aggregation of bits (i.e.,
when not considering quantized inputs that require scale mul-
tiplication and hence without SepAgg [14] being applica-
tble) to Prio+ [2]. For a fair comparison, we translate the ap-
proach Prio+ [2] in our three party dishonest-majority setting.
As we can see, even for exact bit-to-arithmetic conversion, we
improve over Prio+ by factor 2.4× for n = 10^2. When appl-
ying our approximate bit-to-arithmetic conversion (cf. 3.1),
this improvement increases to a factor of 4×.

### B.3 ScionFL-Aura: Additional Details

Here, we provide additional details of ScionFL-Aura.

#### B.3.1 Sub-protocols

In this section, we give the sub-protocols for ScionFL-Aura (cf. §5.1.1). Note that for sake of simplicity, we do not include optimizations discussed in §5.1.3.

Alg. 2 computes the aggregation of α quantized vectors. As shown in Eq. 4, the dequantized value of a vector \( \mathbf{Y} \), given its quantized form \((\overline{\mathbf{y}}, s_y, s_y')\), can be computed as

\[
\mathbf{Y} = s_y \oplus \overline{\mathbf{y}} \circ (s_y' - s_y) .
\]

The above operation essentially places \( s_y \) in those positions of the vector \( \mathbf{Y} \) with the corresponding bit in \( \overline{\mathbf{y}} \) being zero, and the rest with \( s_y' \).

#### Algorithm 2 Quantized Aggregation

1: procedure AGGREGATE\(((\overline{\mathbf{y}}_i, s_y^\text{min}_i, s_y^\text{max}_i))_{i \in \alpha})
2: \[\mathbf{z} \leftarrow 0\]
3: for \( k \leftarrow 1 \) to \( \alpha \) do
4: \[\mathbf{z} \leftarrow \mathbf{z} + (s_y^\text{min}_k \oplus \overline{\mathbf{y}}_k \circ (s_y^\text{max}_k - s_y^\text{min}_k))\]
5: end for
6: \[\mathbf{z} \leftarrow \mathbf{z}/\alpha\]
7: return \(\mathbf{z}\)
8: end procedure

Alg. 3 computes the L2-norm of a quantized vector. As discussed in §2.4, a quantized vector \( \overline{\mathbf{y}}_\alpha \) consists of a binary vector \( \overline{\mathbf{y}} \) and the respective min. and max. scales \( s_y^\text{min} / s_y^\text{max} \). In this case, we observe that the squared L2-norm can be obtained by first counting the number of zeroes and ones in the vector, denoted by \( N_2 \) and and \( N_0 \) respectively, followed by multiplying them with the square of the respective scales and adding the results, i.e. \( N_2 \cdot (s_y^\text{min})^2 + N_0 \cdot (s_y^\text{max})^2 \). Furthermore, computing the number of ones \( N_2 \) corresponds to the bit-aggregation of the vector \( \mathbf{Y} \), for which our aggregation methods discussed in §4.1 can be utilized.

#### Algorithm 3 L2-Norm Computation (Quantized)

1: procedure L2-NORMQ(\(\overline{\mathbf{y}}, s_y^\text{min}, s_y^\text{max}\))
2: \(\beta \leftarrow \text{LEN}(\overline{\mathbf{y}})\) // Dimension of \(\overline{\mathbf{y}}\)
3: \(N_O \leftarrow \text{SUM}(\overline{\mathbf{y}})\) // Number of ones in \(\overline{\mathbf{y}}\)
4: \(N_Z \leftarrow \beta - N_O\) // Number of zeros in \(\overline{\mathbf{y}}\)
5: return \(\sqrt{N_Z \cdot (s_y^\text{min})^2 + N_O \cdot (s_y^\text{max})^2}\)
6: end procedure

Alg. 4 is used to compute the cosine distance between a quantized vector \( \mathbf{Y} \) and a reference vector \( \mathbf{S} \). The cosine distance is given by \(\mathbf{y} \odot \mathbf{S} \), where \(||\cdot||\) corresponds to the L2-norm of the input vector. Using Eq. 4, we can write

\[
\mathbf{y} \odot \mathbf{S} = (s_y^\text{min} \oplus \overline{\mathbf{y}} \circ (s_y^\text{max} - s_y^\text{min})) \circ \mathbf{S} = s_y^\text{min} \odot \mathbf{S} \circ \mathrm{SUM}(\overline{\mathbf{y}} \odot \mathbf{S}) .
\]

Thus, the inner product computation of \( \mathbf{y} \odot \mathbf{S} \) reduces to computing \(\overline{\mathbf{y}} \odot \mathbf{S} \), followed by two multiplications.

#### Algorithm 4 Cosine Distance Calculation

1: procedure COSINEQ(\(\overline{\mathbf{y}}, s_y^\text{min}, s_y^\text{max}\), \(\mathbf{S}\))
2: \(L_2^\mathbf{y} \leftarrow \text{L2-NORMQ}(\overline{\mathbf{y}}, s_y^\text{min}, s_y^\text{max})\)
3: \(L_2^\mathbf{S} \leftarrow ||\mathbf{S}||\) // Computes L2-norm
4: \(\alpha \leftarrow \text{SUM}(\overline{\mathbf{y}})\) // Sum of elements of \(\overline{\mathbf{y}}\)
5: \(\beta \leftarrow \text{INNER-PRODUCT}(\overline{\mathbf{y}}, \mathbf{S})\)
6: \(\gamma = s_y^\text{min} \odot \mathbf{S} \circ \mathrm{SUM}(\overline{\mathbf{y}} \odot \mathbf{S})\)
7: return \(\gamma/(L_2^\mathbf{y} \cdot L_2^\mathbf{S})\)
8: end procedure

#### B.3.2 Evaluation on VGG11

In addition to our results in §5.1.2, we evaluate the Min-Max attack on VGG11 trained with CIFAR10. The experimental setup is identical to §5.1.2. The results are shown in Fig. 18.

Similarly as for ResNet9 (cf. Fig. 10), the Min-Max attack substantially reduces the validation accuracy when training VGG11: We observe drops of up to 36.8%. However, on average, VGG11 is less impacted by the attack. Concretely, only 15% of the iterations observe a validation accuracy reduction of about 15% or more when using no compression. One third of the training rounds are impacted by about 15% or more when using Kashin’s representation (KSQ) while with the Hadamard transform (HSQ) only very few training rounds showed a significant accuracy reduction. Thus, HSQ seems to be inherently more robust against targeted poisoning.
With ScionFL-Aura, the accuracy reduction is still noticeable smaller for all variants. With HSQ, on average 0.28 malicious updates are included in global updated instead of 2.24 without defense. With respect to the validation accuracy, the difference between having no attack and employing ScionFL-Aura when under attack is less than 4% in almost all training iterations. When using KSQ, a global update includes just 0.44 malicious updates on average, and the attack impact is at least halved in two third of the training iterations.

### B.3.3 Comparison to FLAME [97]

Our untargeted poisoning defense ScionFL-Aura has components with similarities to the backdoor defense FLAME [97], which has three steps: density-based clustering, clipping, and noise addition. However, there are important differences, which we emphasize in the following.

1. **Magnitude Boundary.** FLAME’s clipping is done with respect to the median Euclidean distance of the local updates to the previous global model. However, especially with non-iid data and in early training phases, each training iteration may exhibit significant differences even for consecutive iterations. Hence, using the recent average norm (assuming the majority of updates is benign) as in ScionFL-Aura (cf. Step 1 in §5.1.1 and Lines 3-14 in Alg. 1) intuitively gives a better estimation for a benign magnitude in the current training state.

2. **Filtering.** FLAME compares cosine similarity in a pair-wise fashion among individual updates, i.e., it computes \(\frac{n(n-1)}{2}\) cosine distances per iteration while ScionFL-Aura does \(n\) (cf. Step 2 in §5.1.1 and Line 16 in Alg. 1). While ScionFL-Aura sorts local updates based on cosine similarity (cf. Step 2 in §5.1.1 and Line 18 in Alg. 1), FLAME uses secure clustering with low cubic complexity [25]. FLAME only accepts updates assigned to the largest cluster, which can lead to an exclusion of benign updates and thus significantly slowing down the training by removing valuable benign contributions. In contrast, ScionFL-Aura removes only a fixed number of updates, thereby enabling an efficient trade-off that reduces the attack’s effect to a tolerable level (even if a few malicious contributions are not filtered out) with a low false positive rate.

3. **Differential Privacy:** After the clipping, FLAME aggregates the updates and adds noise in the cleartext to create a differentially private new global model. We do not consider differential privacy in our work, however, the noise addition can trivially be added to our system.

For more details on FLAME, we refer the reader to [97].