ScionFL: Efficient and Robust Secure *Quantized* Aggregation

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

* VMware Research Group † Technical University of Darmstadt ‡ Aptos Labs and Bar-Ilan University
§ Technology Innovation Institute ¶ DFINITY Foundation || Royal Holloway, University of London

***Abstract*—Secure aggregation is commonly used in federated learning (FL) to alleviate privacy concerns related to the central aggregator seeing all parameter updates in the clear. Unfortunately, most existing secure aggregation schemes ignore two critical orthogonal research directions that aim to (i) significantly reduce client-server communication and (ii) mitigate the impact of malicious clients. However, both of these additional properties are essential to facilitate cross-device FL with thousands or even millions of (mobile) participants.**

In this paper, we unite both research directions by introducing ScionFL, the first secure aggregation framework for FL that operates *efficiently* on quantized inputs and simultaneously provides robustness against malicious clients. Our framework leverages (novel) multi-party computation (MPC) techniques and supports multiple linear (1-bit) quantization schemes, including ones that utilize the randomized Hadamard transform and Kashin's representation.

Our theoretical results are supported by extensive evaluations. We show that with *no overhead for clients* and moderate overhead for the server compared to transferring and processing quantized updates in plaintext, we obtain comparable accuracy for standard FL benchmarks. Moreover, we demonstrate the robustness of our framework against state-of-the-art poisoning attacks.

I. Introduction

Federated learning (FL) [94] is a paradigm for large-scale distributed machine learning, where in each training round a subset of clients locally updates a global model that is then centrally aggregated. FL quickly gained popularity 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 [95], [109], [57], [120], [132]. Hence, many secure aggregation schemes, e.g., [26], [54], [99], have been developed, where the aggregator only learns the aggregation result, i.e., the global model (we refer to [69], [103] for a discussion of differential privacy in FL as an orthogonal privacy-enhancing paradigm). Most prominently, in the "SecAgg" protocol [26], 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 [137], [68], [84]). Moreover, recent research has shown that a *single malicious aggregator* can reconstruct individual training data from clients' inputs, despite the use of secure aggregation [24], [104], [25].

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 [94]) 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., [126], [31], [129], [4], [12], [18], [117].

Unfortunately, so far the ML community has worked on optimizing FL for resource efficiency, while the security community has separately worked on optimizing secure aggregation for privacy. One of the very few exceptions is a work [36] that combines SecAgg with sketching [114] to compress gradients. Besides the mentioned disadvantages of SecAgg, their work considers only one compression method. However, because of the trade-offs between accuracy and computational efficiency offered by various forms of quantization, which become particularly relevant for gradients with millions of coordinates, it is important to adopt a modular approach.

Beyond data privacy threats, FL was also shown to be vulnerable to manipulations by malicious clients who alter their local models/updates, affecting the characteristics of the final global model [119], [52], [136], [10], [20], [118]. This highlights the need for effective *defense measures* capable of thwarting such attacks. While there exists a plethora of such attacks and defenses in the FL literature (see §A-F), 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 [52], [118], [11]. Unfortunately, existing defenses for such attacks cannot be efficiently translated to secure aggregation in the quantized setting [102], [141].

A. Our Contributions

We propose ScionFL, a framework that enables *efficient* and *robust* secure aggregation for FL with a distributed aggregator that operates *directly* on *quantized* parameter updates. Specifically, ScionFL has virtually *no additional communication overhead for clients* compared to the insecure transfer of quantized plaintext updates. Achieving this is non-trivial for prior single-server solutions that are based on masking techniques or homomorphic encryption (HE) [26], [140].

In ScionFL, we use outsourced multi-party computation (MPC), where the clients apply computationally efficient secret-sharing techniques to distribute their sensitive (quantized) parameter updates among multiple servers that together form a distributed aggregator. These servers then use MPC protocols [41], [42], [17] to securely compute the aggregation and only reveal the updated global model.

The distributed aggregator model is well established [54], [90], [110], [58] (although prior works cannot efficiently handle quantized inputs), offering practical benefits over single-server solutions, such as not requiring an interactive setup between clients [102], [99] and efficient dropout handling. Moreover, in recent years, numerous studies have demonstrated that single server aggregation methods are susceptible to privacy attacks when an aggregator is corrupt [121], [24], [104], [25], [55], [135]. The vulnerability of these methods stems from the fact that the aggregator holds complete authority in selecting clients and the data transmitted to and received from them. On the other hand, with our MPC protocols, data privacy can be guaranteed even if all servers except one are compromised or their operators receive subpoena requests.

As MPC protocols typically induce significant overhead in terms of (inter-server) communication, we propose optimizations for secure aggregation that *support any "linear" quantization scheme*, including *1-bit quantization* schemes that uses preprocessing like random rotations [126] and Kashin's representation [31]. We also study novel *approximate* MPC variants that leverage FL's noise tolerance and might be of independent interest. We formally prove that our resultant secure aggregation scheme is an *unbiased estimator* of the arithmetic mean and explore efficiency-accuracy trade-offs.

We implement a combined end-to-end FL evaluation and MPC simulation environment. Our prototype implementation allows to assess the performance and accuracy of our solution for stochastic quantization schemes, including recent state-of-the-art distributed mean estimation techniques [126], [31]. Our results demonstrate that standard FL benchmarks' accuracy is barely impacted while our optimizations and approximation can significantly reduce inter-server communication. For example, when training the LeNet architecture for image classification on the MNIST data set [83] using 1-bit stochastic quantization with Kashin's representation [31], training accuracy is only slightly reduced from 99.04% to 98.71% after 1000 rounds, while inter-server communication drops from 16.14 GB to 0.94 GB compared to naive MPC-based secure aggregation when considering 500 clients per round, an improvement by factor 17.2×.

Since clients may act maliciously and try to degrade accuracy with their updates, we design a novel bipartite defense mechanism called ScionFL-Aura to ensure the robustness of our framework. Specifically, we provide protection against state-of-the-art untargeted poisoning attacks [118], combining magnitude clipping and directional filtering based on the gradients' approximate L2-norms and cosine distances. Notably, ScionFL-Aura is the first defense mechanism to work directly on quantized inputs and thus enables a highly efficient realization in MPC, whereas existing works require expensive MPC conversions of all individual parameters [5], [102]. We summarize our contributions as follows:

1) First *secure aggregation* framework called ScionFL to consider (1-bit) quantization with almost *no communication overhead for clients* compared to *plaintext quantized FL.*

2) Novel *optimizations and approximations* to reduce MPC-induced inter-server communication, with performance/accuracy trade-offs.

3) End-to-end FL evaluation and MPC simulation environment, demonstrating the efficiency and accuracy of ScionFL.

4) First efficient and effective FL (poisoning) defense operating directly on quantized updates.

Though we study FL as our primary application, our secure aggregation protocols have numerous other applications like privacy-preserving aggregate statistics, for which there are currently (less efficient) real-world deployments that also rely on distributed aggregators (e.g., telemetry reporting in Mozilla's Firefox browser [1], [2]). For these settings, we achieve improvements in communication of up to 4× over prior works like Prio+ [2] and the details are provided in §III-E.

B. Related Work

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

Quantization and Compression in FL: In this work, we focus on quantization to reduce communication in FL. However, an alternative line of work investigates compression techniques for the same purpose. There are three main directions for gradient compression in cross-device FL: (i) gradient sparsification (e.g., [53], [123], [3], [76]), (ii) client-side memory and error-feedback (e.g., [117], [4], [112], [19]), and (iii) entropy encodings (e.g., [126], [128], [4]). Reviews of current state-of-the-art gradient compression techniques and some open challenges can be found in [69].

Compression techniques are less suited for the requirements of secure aggregation for cross-device FL than quantization, e.g., due to computational overhead, incompatibility with secure aggregation, and state-requirements on the client side. We discuss more details in §II-B and §A-B.

Secure Aggregation & Model Inference Attacks: In conventional FL with a single aggregator, clients share locally trained model updates with a central party to train a global model. However, sharing those updates makes the system vulnerable to data leakage. Attacks exploiting this leakage are called *inference attacks* [23], [82], [101]. Even a semi-honest central server can learn confidential information about the used private training data by analyzing the received local model updates.

A common countermeasure against inference attacks is to use secure aggregation [50], [81] (cf. §A-E). 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 [26], [54], [16], [116], [68], [122]. 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, SecAgg [26], combines secret sharing with authenticated symmetric encryption, but requires 4 communication rounds per training iteration among servers and client. Bell et al. [16] improve upon SecAgg [26] by reducing client communication and computation to poly-logarithmic complexity. However, from a practical point of view also [16] as well as other existing secure aggregation protocols designed for FL still exhibit significant computation and communication overhead: Due to underlying secret sharing or encryption, those protocols typically encode each local update in 32-bit and add computational overhead caused by the required cryptographic operations. In contrast, ScionFL enables highly communication-efficient secure aggregation thanks to 1-bit quantization and causes almost *no* additional communication overhead on the client side compared to plaintext FL. Fereidonni et al. [54] provide more details by comparing several secure aggregation protocols with respect to efficiency and practicality. Mansouri et al. [90] provide a comprehensive analysis of secure aggregation schemes w.r.t. their suitability for FL.

To the best of our knowledge, only Chen et al. [36] and Beguier et al. [15] have considered both compression and secure aggregation in combination for FL. Specifically, Chen et al. [36] combine SecAgg [26] with sparse random projection and count-sketching [114] for compression. Moreover, they add noise using a distributed discrete Gaussian mechanism to generate a differential private output. Beguier et al. [15] combine arithmetic secret-sharing with Top-k sparsification [123] and 1-bit quantization [18]. As we point out in §A-B, both sketching and sparsification are sub-optimal 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.

Poisoning Attacks & Defenses: FL was shown vulnerable to manipulations by malicious clients [10], [11], [52], [118], [142]. Targeted or backdoor attacks aim at manipulating the inference results for specific attacker-chosen inputs [10], [142], while untargeted poisoning attacks [11], [52], [118] reduce the overall global model accuracy. As untargeted attacks are considered to be more severe (given they are harder to detect, cf. §IV), we focus on defending those using Byzantine-robust defenses like [141], [102]. In §A-F, we provide a more detailed overview of poisoning attacks and defenses.

Comparison: We compare ScionFL qualitatively in Tab. I to the state-of-the-art related work with respect to aggregation and quantization, as well as robustness against poisoning.

Categories	Reference	Technique	M.P.	Quant.	P.R.	Dist. Servers	No Client Interaction
Aggregation	[39]	MPC	✓	✗	✗	✓	✓
	[16]	Masking	✓	✗	✗	✗	✗
Quantization	[126]	–	✗	✓	✗	✗	✓
	[15]	MPC	✓	✗	✗	✓	✓
	[36]	Masking+DP	✓	✓	✗	✗	✗
Robustness	[102]	MPC	✓	✗	✓	✓	✓
	[141]	–	✗	✗	✓	✗	✓
ScionFL	**This**	MPC	✓	✓	✓	✓	✓

TABLE I: High-level comparison of ScionFL and previous works. Notation: MPC—Secure Multi-Party Computation, DP—Differential Privacy, M.P.—Model Privacy, Quant.—Quantization (refers to the schemes where compressed gradients are communicated by the clients to the aggregator(s)), P.R.—Poisoning Resilience, Dist.—Distributed. Client Interaction refers to interaction among clients. Since the body of literature is vast, comparison is made against only a subset representing each category.

II. Problem Statement

We now define the precise problem of secure quantized aggregation for FL, which we address in our work. 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.

A. Aggregation for Federated Learning

Google introduced federated learning (FL) as a distributed machine learning (ML) paradigm in 2016 [76], [94]. 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^t.

2) Each selected client C_i, $i \in [\mathsf{n}]$, sets its local model $w_i^{t+1} = G^t$ and improves it using its local dataset D_i for E epochs (i.e., local optimization steps):

$$w_i^{t+1} \leftarrow w_i^{t+1} - \eta_{\mathsf{C}_i} \frac{\partial L(w_i^{t+1}, B_{i,e})}{\partial w_i^{t+1}}, \tag{1}$$

where L is a loss function, η_{C_i} is the clients' learning rate, and $B_{i,e} \subseteq D_i$ is a batch drawn from D_i in epoch e, where $e \in [E]$. 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 f_{agg}:

$$G^{t+1} \leftarrow G^t - \eta_{\mathsf{S}} \cdot f_{agg}(w_1^{t+1}, \ldots, w_{\mathsf{n}}^{t+1}), \tag{2}$$

where η_{S} is the server's learning rate. The most commonly used aggregation rule, which we also focus on, is FedAvg [94]. It averages the local updates as follows:

$$\mathsf{FedAvg}(w_1^{t+1}, \ldots, w_{\mathsf{n}}^{t+1}) = \sum_{i=1}^{\mathsf{n}} \frac{|D_i|}{\mathsf{n}} w_i^{t+1} \tag{3}$$

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

B. Stochastic Quantization

Quantization is a central building block in FL, where data transmission 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 the desired properties and constructions of quantization schemes that will play a key role in our system design and additional details are provided in §A-A.

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 $w \in \mathbb{R}^d$, being unbiased means that $\mathbb{E}[\hat{w}] = w$. Unbiasedness is desired because it guarantees that the mean squared error (MSE) of the mean's estimation 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., [94], [86], [70]), the MSE measure (or normalized MSE, a.k.a. NMSE, cf. §III-D) is indeed the quantity of interest since it affects the convergence rate and often the final accuracy of 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. Using 1-bit quantization has been the focus of many recent works concerning distributed and FL network efficiency (e.g., [129], [18], [127], [117], [65]). 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 $\sigma_i = \mathit{Bernoulli}(\frac{X_i - \mathsf{s}_X^{min}}{\mathsf{s}_X^{max} - \mathsf{s}_X^{min}})$, where $\mathsf{s}_X^{max} = \max(X)$ and $\mathsf{s}_X^{min} = \min(X)$. The coordinate is then reconstructed by the receiver as $\vec{X}_\sigma^i = \mathsf{s}_X^{min} + \vec{\sigma}_X^i \cdot (\mathsf{s}_X^{max} - \mathsf{s}_X^{min})$. This simple technique results in an unbiased quantization as desired, i.e., $\mathbb{E}[\vec{X}_\sigma^i] = \mathbb{E}[\mathsf{s}_X^{min} + \vec{\sigma}_X^i \cdot (\mathsf{s}_X^{max} - \mathsf{s}_X^{min})] = X_i$.

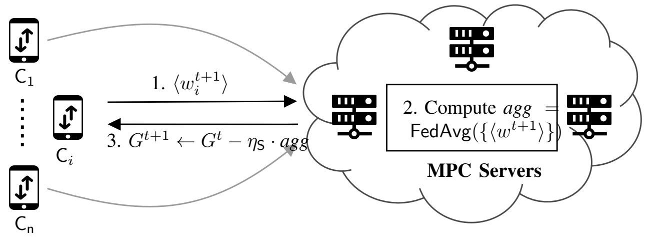
Linear SQ Techniques: A key requirement of being able to perform secure aggregation *efficiently* is being able to aggregate client gradients in their compressed (i.e., quantized) form. The schemes with this property are called "linear" henceforth. One approach involves a "global scales" method, where each client securely learns the maximal and the minimal value across all gradients, i.e., $\mathsf{s}^{max} = \max_c\{\mathsf{s}_{X_c}^{max}\}$ and $\mathsf{s}^{min} = \min_c\{\mathsf{s}_{X_c}^{min}\}$.[1] Despite its simplicity, this method has several drawbacks: (i) it requires a preliminary communication stage, (ii) it reveals the global extreme values (even if they are computed securely across all clients), and (iii) 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 [15] and adapt it to our setting (cf. §III-A).

While we focus on the mentioned vanilla SQ, SQ with random rotation [126] and SQ with Kashin's representation [31], [88], [115], our framework seamlessly supports any "linear" quantization scheme, namely, any quantization technique that allows for aggregation in a compressed form (cf. §C-A).

Other Approaches: We acknowledge recent advances in gradient quantization [129], [128], [46], [13]), but these non-linear techniques are less applicable to our framework.

C. 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 [54], [102], [99], [48]: The set of selected FL clients uses 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.



C_1
1. $\langle w_i^{t+1} \rangle$
C_i
3. $G^{t+1} \leftarrow G^t - \eta_{\mathsf{S}} \cdot agg$
C_n
2. Compute $agg = \mathsf{FedAvg}(\{\langle w^{t+1} \rangle\})$
MPC Servers

Fig. 1: Secure aggregation in FL for n clients using outsourced MPC; $\langle w_i^{t+1} \rangle$ denotes the secret-shares of gradient w_i^{t+1} that client C_i has in round $t+1$; G^t is the model of the previous round and η_{S} the server-side learning rate.

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.

MPC 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 transcripts. This well-established assumption, both in theory [7], [67], [96], [102], [105], [22], [62] and in practice [1], [130], [6], 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 monitors or

[1] This approach resembles "scaler sharing" in TernGrad [134].

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 $\mathcal{A}$ corrupts all except one of the involved servers. Furthermore, our framework is easily extendable to provide *malicious privacy*, which ensures input privacy even when the corrupted servers try to actively cheat [110].

Malicious Clients: We anticipate that some clients might behave maliciously to negatively affect the quality of the global model with manipulated updates (i.e., poisoning attacks, cf. §IV). This is because 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. We assume an honest majority of clients as is standard in FL, yet our framework is secure even when malicious clients collude with corrupted servers.

Preprocessing Model: We use MPC in the preprocessing model [45], [43], [14], [44]. 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 time before the actual computation. This gives several advantages, e.g., service providers can exploit cheap spot instances. It also 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 [7], [56], [34], [97], [78] and in ML systems [126], [12] to optimize communication.

D. 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 $\vec{X} = \{x_1, \ldots, x_{\mathsf{m}}\}$ comprising of ℓ-bit values will be quantized to obtain a triple $\vec{X}_\sigma = (\vec{\sigma}_X, \mathsf{s}_X^{min}, \mathsf{s}_X^{max})$. Here, $\vec{\sigma}_X$ is an m-dimensional binary vector with a zero at an index indicating the value s_X^{min} and a one indicating the value s_X^{max}. Also, s_X^{min} and s_X^{max} correspond to the minimum and maximum values in the vector $\vec{X}$. With this binary quantization (cf. §II-B), the quantized value at the ith dimension, denoted by $\vec{X}_\sigma^i$, can be written as

$$\vec{X}_\sigma^i = \mathsf{s}_X^{min} + \vec{\sigma}_X^i \cdot (\mathsf{s}_X^{max} - \mathsf{s}_X^{min}). \tag{4}$$

Before going into the details of aggregation, we provide some of the basic notation that will be utilized throughout the paper.

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

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

$$\mathbf{M}_i^j = \mathbf{U}_i^j \cdot \mathbf{V}_i^1 \quad \text{and} \quad \mathbf{N}_i^j = \mathbf{U}_i^j + \mathbf{V}_i^1, \quad \text{where } i \in [\alpha], j \in [\beta].$$

Quantized Aggregation: To perform aggregation on quantized inputs, a set of n clients first locally prepares their quantized triples, $(\vec{\sigma}_X, \mathsf{s}_X^{min}, \mathsf{s}_X^{max})$, corresponding to their locally trained ML model updates (i.e., gradients) and submits these 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 $(\vec{\mathbf{B}}_{\mathsf{n}\times\mathsf{m}}, \vec{\mathbf{U}}_{\mathsf{n}\times 1}, \vec{\mathbf{V}}_{\mathsf{n}\times 1})$. Here, $\vec{\mathbf{B}}$ is a binary matrix that corresponds to the $\vec{\sigma}_X$ vector of the clients. Similarly, $\vec{\mathbf{U}}$ and $\vec{\mathbf{V}}$ correspond to the s_X^{min} and s_X^{max} values of the above-mentioned triple. The quantized aggregate is defined as

$$\vec{\mathbf{X}}_{1\times\mathsf{m}} = \mathsf{Agg\text{-}R}\left(\vec{\mathbf{U}}_{\mathsf{n}\times 1} \;\oplus\; \vec{\mathbf{B}}_{\mathsf{n}\times\mathsf{m}} \circ (\vec{\mathbf{V}}_{\mathsf{n}\times 1} - \vec{\mathbf{U}}_{\mathsf{n}\times 1})\right). \tag{5}$$

Ideal Functionality: To perform secure aggregation of quantized updates using MPC, we model the aggregation as an ideal functionality $\mathcal{F}_{\mathsf{SecAgg}}$ (Fig. 2). We consider a set of τ servers to which the clients secret-share their quantized updates. 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.

Functionality $\mathcal{F}_{\mathsf{SecAgg}}$

$\mathcal{F}_{\mathsf{SecAgg}}$ interacts with all the τ servers in $\mathcal{S}$ and an adversary $\mathcal{A}$ that controls a subset of the servers in $\mathcal{S}$.

Input: $\mathcal{F}_{\mathsf{SecAgg}}$ receives $\langle\cdot\rangle$-shares of the matrix triple $(\vec{\mathbf{B}}_{\mathsf{n}\times\mathsf{m}}, \vec{\mathbf{U}}_{\mathsf{n}\times 1}, \vec{\mathbf{V}}_{\mathsf{n}\times 1})$ from the honest servers in $\mathcal{S}$, while the adversary $\mathcal{A}$ provides the $\langle\cdot\rangle$-shares on behalf of the corrupt servers. Here $\mathbf{B}_i^j \in \{0, 1\}$ and $\mathbf{U}_i^j, \mathbf{V}_i^j \in \mathbb{Z}_{2^\ell}$.

Computation: $\mathcal{F}_{\mathsf{SecAgg}}$ reconstructs $(\vec{\mathbf{B}}, \vec{\mathbf{U}}, \vec{\mathbf{V}})$ from its $\langle\cdot\rangle$-shares.

- Set $\vec{\mathbf{S}}_{\mathsf{n}\times 1} = \vec{\mathbf{V}}_{\mathsf{n}\times 1} - \vec{\mathbf{U}}_{\mathsf{n}\times 1}$ and compute $\vec{\mathbf{W}}_{\mathsf{n}\times\mathsf{m}} = \vec{\mathbf{B}}_{\mathsf{n}\times\mathsf{m}} \circ \vec{\mathbf{S}}_{\mathsf{n}\times 1}$.
- Compute $\vec{\mathbf{X}}_{\mathsf{n}\times\mathsf{m}} = \vec{\mathbf{U}}_{\mathsf{n}\times 1} \oplus \vec{\mathbf{W}}_{\mathsf{n}\times\mathsf{m}}$.
- Compute $\vec{\mathbf{Y}}_{1\times\mathsf{m}} = \mathsf{Agg\text{-}R}(\vec{\mathbf{X}}_{\mathsf{n}\times\mathsf{m}})$, i.e., $\vec{\mathbf{Y}}^j = \sum_{i=1}^{\mathsf{n}} \mathbf{X}_i^j$ for $j \in [\mathsf{m}]$.

Output: $\mathcal{F}_{\mathsf{SecAgg}}$ computes $\langle\cdot\rangle$-shares of $\vec{\mathbf{Y}}$ and sends the respective shares to the servers in $\mathcal{S}$.

Fig. 2: Ideal functionality for semi-honest secure quantized aggregation for linear stochastic binary quantization.

III. Our Framework: ScionFL

We now detail our framework "ScionFL" from an MPC standpoint, covering the sharing semantics, client interaction with MPC servers, and secure aggregation of client updates. Our generic constructions utilize MPC in a black-box fashion [45], [41], [42], [17], [113], however, the full MPC protocols, including inner product computation, multiplication, and bit-to-arithmetic conversion, are detailed in §B-A.

Masked Evaluation: In our MPC protocols, we use the masked evaluation technique [60], [17], [35], [93], [125],

[78], which enables costly *data-independent* calculations to be completed in a preprocessing phase, thus enabling a fast and efficient *data-dependent* online phase (cf. §II-C). In this model, the secret-share of every element $\mathsf{v} \in \mathbb{Z}_{2^\ell}$, denoted by $\langle \mathsf{v} \rangle$, is associated with two values: a random mask $\lambda_\mathsf{v} \in \mathbb{Z}_{2^\ell}$ and a masked value $\mathsf{m}_\mathsf{v} \in \mathbb{Z}_{2^\ell}$ such that $\mathsf{v} = \mathsf{m}_\mathsf{v} + \lambda_\mathsf{v}$. While λ_v is split and distributed as q shares as per the underlying MPC scheme (cf. §B-A), m_v is given to all MPC servers.[2] Since λ_v is random and independent of v, all operations involving λ_v values alone can be computed during the preprocessing phase and thereby leading to a fast online phase.

Client-Server Interaction: Before going into the details of aggregation among τ 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 $\mathsf{v} \in \mathbb{Z}_{2^\ell}$ owned by client C, it first non-interactively computes the additive shares of the mask λ_v using the shared randomness setup discussed in §II-C. The masked value is then computed as $\mathsf{m}_\mathsf{v} = \mathsf{v} - \lambda_\mathsf{v}$ 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.[3] For Boolean values (i.e., in $\mathbb{Z}_2$) the procedure is similar except that addition/subtraction is replaced with XOR and multiplication with AND. We use $\langle \cdot \rangle^\mathbf{B}$ 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 $\mathcal{F}_{\mathsf{SecAgg}}$ functionality specified in Fig. 2 and obtain the aggregated vectors in $\langle \cdot \rangle$-shared form. Recall from §II-A 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 interacts solely with a single server (apart from a one-time shared-randomness setup), as in the privacy-free variant with a single parameter server [94].

A. MPC-based Aggregation

We now discuss three approaches for instantiating $\mathcal{F}_{\mathsf{SecAgg}}$ 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 $\vec{\mathbf{U}}_{\mathsf{n}\times 1}$ and $\vec{\mathbf{V}}_{\mathsf{n}\times 1}$ along with the $\langle \cdot \rangle^\mathbf{B}$-shares of $\vec{\mathbf{B}}_{\mathsf{n}\times\mathsf{m}}$.

Approach I: A naive instantiation of $\mathcal{F}_{\mathsf{SecAgg}}$ would be to have the servers convert binary shares of the matrix $\vec{\mathbf{B}}$ to their arithmetic shares first, as in Prio+ [2]. This is possible in MPC with a bit-to-arithmetic conversion protocol Π_{BitA} [97], [107], [2], [78], which computes the arithmetic shares of $\mathsf{b} \in \mathbb{Z}_2$ from its Boolean sharing. Once the arithmetic shares are generated, the servers can use the inner-product protocol Π_{IP} on each of the m columns of matrix $\vec{\mathbf{B}}$ with the locally computed column vector $(\vec{\mathbf{V}}_{\mathsf{n}\times 1} - \vec{\mathbf{U}}_{\mathsf{n}\times 1})$ to obtain the row aggregate. They complete the protocol by adding a row aggregate of $\vec{\mathbf{U}}$ to each column of the matrix computed in the previous step. The formal protocol $\Pi^{\mathrm{I}}_{\mathsf{SecAgg}}$ is given in Fig. 3.

Protocol $\Pi^{\mathrm{I}}_{\mathsf{SecAgg}}(\langle \vec{\mathbf{B}}_{\mathsf{n}\times\mathsf{m}} \rangle^\mathbf{B}, \langle \vec{\mathbf{U}}_{\mathsf{n}\times 1} \rangle, \langle \vec{\mathbf{V}}_{\mathsf{n}\times 1} \rangle)$

1. Locally compute $\langle \vec{\mathbf{S}}_{\mathsf{n}\times 1} \rangle = \langle \vec{\mathbf{V}}_{\mathsf{n}\times 1} \rangle - \langle \vec{\mathbf{U}}_{\mathsf{n}\times 1} \rangle$.
2. Compute $\langle \vec{\mathbf{B}} \rangle = \Pi_{\mathsf{BitA}}(\langle \vec{\mathbf{B}} \rangle^\mathbf{B})$.
3. Compute $\langle \vec{\mathbf{W}}^j \rangle = \Pi_{\mathsf{IP}}(\langle \vec{\mathbf{B}}^j \rangle, \langle \vec{\mathbf{S}} \rangle)$, for each $j \in [\mathsf{m}]$.
4. Locally compute $\langle \vec{\mathbf{Z}}_{1\times 1} \rangle = \mathsf{Agg\text{-}R}(\langle \vec{\mathbf{U}}_{\mathsf{n}\times 1} \rangle)$.
5. Locally compute $\langle \vec{\mathbf{Y}}_{1\times\mathsf{m}} \rangle = \langle \vec{\mathbf{Z}}_{1\times 1} \rangle \oplus \langle \vec{\mathbf{W}}_{1\times\mathsf{m}} \rangle$.

Fig. 3: Secure aggregation – Approach I.

In Fig. 3, $\Pi^{\mathrm{I}}_{\mathsf{SecAgg}}$ invokes Π_{BitA} for each bit in matrix $\vec{\mathbf{B}}$, resulting in $\mathsf{n} \cdot \mathsf{m}$ invocations. Using the masked evaluation technique [79], [125], the online communication of the inner product protocol Π_{IP} can be made independent of the dimension n, which in our case corresponds to the number of clients.

Approach II: We use the bit injection protocol [97], [107], [77], denoted by Π_{BI}, which computes the arithmetic sharing of $\mathsf{b} \cdot \mathsf{s}$ given the Boolean sharing of $\mathsf{b} \in \mathbb{Z}_2$ and the arithmetic sharing of $\mathsf{s} \in \mathbb{Z}_{2^\ell}$. Given $\langle \vec{\mathbf{M}}_{\alpha\times 1} \rangle^\mathbf{B}$ and $\langle \vec{\mathbf{N}}_{\alpha\times 1} \rangle$, the high-level idea is to effectively combine the Π_{BitA} and Π_{IP} protocol to a slightly modified instance of Π_{BI} that directly computes the sum [79], [125] (denoted by $\sum_{i=1}^{\alpha} \mathbf{M}_i^1 \cdot \mathbf{N}_i^1$) instead of the individual positions. This can be achieved following Eq. (16) and the details appear in Fig. 17 in §B-A. One significant advantage of this technique is that the overall online communication is no longer dependent on the number of clients n. $\Pi^{\mathrm{II}}_{\mathsf{SecAgg}}$ denotes the resulting protocol and the formal details are given in Fig. 4.

Protocol $\Pi^{\mathrm{II}}_{\mathsf{SecAgg}}(\langle \vec{\mathbf{B}}_{\mathsf{n}\times\mathsf{m}} \rangle^\mathbf{B}, \langle \vec{\mathbf{U}}_{\mathsf{n}\times 1} \rangle, \langle \vec{\mathbf{V}}_{\mathsf{n}\times 1} \rangle)$

1. Locally compute $\langle \vec{\mathbf{S}}_{\mathsf{n}\times 1} \rangle = \langle \vec{\mathbf{V}}_{\mathsf{n}\times 1} \rangle - \langle \vec{\mathbf{U}}_{\mathsf{n}\times 1} \rangle$.
2. Compute $\langle \vec{\mathbf{W}}^j \rangle = \Pi_{\mathsf{BI}}(\langle \vec{\mathbf{B}}^j \rangle^\mathbf{B}, \langle \vec{\mathbf{S}} \rangle)$, for each $j \in [\mathsf{m}]$.
3. Locally compute $\langle \vec{\mathbf{Z}}_{1\times 1} \rangle = \mathsf{Agg\text{-}R}(\langle \vec{\mathbf{U}}_{\mathsf{n}\times 1} \rangle)$.
4. Locally compute $\langle \vec{\mathbf{Y}}_{1\times\mathsf{m}} \rangle = \langle \vec{\mathbf{Z}}_{1\times 1} \rangle \oplus \langle \vec{\mathbf{W}}_{1\times\mathsf{m}} \rangle$.

Fig. 4: Secure aggregation – Approach II.

Approach III using SepAgg: In a closely related work [15], the authors combine the SIGNSGD compression technique of [18] 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 a non-optimal communication overhead of factor $\log_2 2\mathsf{n}$, where n is the number of involved participants each round.

However, the authors of [15] introduce an interesting approximation called "SepAgg" for computing the averaged inner product between bits and scales:

$$\frac{1}{\mathsf{n}} \sum_{i}^{\mathsf{n}} \vec{\sigma}_X^i \cdot \vec{\mathsf{s}}_X^i \approx \frac{1}{\mathsf{n}^2} \left(\sum_{i}^{\mathsf{n}} \vec{\sigma}_X^i \right) \left(\sum_{i}^{\mathsf{n}} \vec{\mathsf{s}}_X^i \right). \qquad (6)$$

[2] Due to differences in the underlying setting, there may be minor differences in how the values m_v and λ_v are distributed among the servers.

[3] If *malicious privacy* is desired, C can send a hash digest of all the m_v values to the remaining MPC servers, who can verify the correctness of messages from S_1.

We adopt the SepAgg method to our setting to compute $\mathsf{Agg\text{-}R}(\vec{\mathbf{B}} \circ (\vec{\mathbf{V}} - \vec{\mathbf{U}}))$ in Eq. (5). In particular, we aggregate the matrices $\vec{\mathbf{B}}$ and $(\vec{\mathbf{V}} - \vec{\mathbf{U}})$ 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^{min} and s_X^{min} across all participants). The formal protocol $\Pi^{\mathrm{III}}_{\mathsf{SecAgg}}$ utilizing SepAgg appears in Fig. 5.

Protocol $\Pi^{\mathrm{III}}_{\mathsf{SecAgg}}(\langle\vec{\mathbf{B}}_{\mathsf{n}\times\mathsf{m}}\rangle^{\mathbf{B}}, \langle\vec{\mathbf{U}}_{\mathsf{n}\times 1}\rangle, \langle\vec{\mathbf{V}}_{\mathsf{n}\times 1}\rangle)$

1. Locally compute $\langle\vec{\mathbf{S}}_{1\times 1}\rangle = \mathsf{Agg\text{-}R}(\langle\vec{\mathbf{V}}_{\mathsf{n}\times 1}\rangle - \langle\vec{\mathbf{U}}_{\mathsf{n}\times 1}\rangle)$.
2. Compute $\langle\vec{\mathbf{T}}^{j}\rangle = \Pi^{\mathsf{sum}}_{\mathsf{BitA}}(\langle\vec{\mathbf{B}}^{j}\rangle^{\mathbf{B}})$, for each $j \in [\mathsf{m}]$.
3. Compute $\langle\vec{\mathbf{W}}^{j}\rangle = \Pi_{\mathsf{Mult}}(\langle\vec{\mathbf{T}}^{j}\rangle, \langle\vec{\mathbf{S}}\rangle)$, for each $j \in [\mathsf{m}]$.
4. Locally compute $\langle\vec{\mathbf{Z}}_{1\times 1}\rangle = \mathsf{Agg\text{-}R}(\langle\vec{\mathbf{U}}_{\mathsf{n}\times 1}\rangle)$.
5. Locally compute $\langle\vec{\mathbf{Y}}_{1\times\mathsf{m}}\rangle = \langle\vec{\mathbf{Z}}_{1\times 1}\rangle \oplus \frac{1}{\mathsf{n}} \cdot \langle\vec{\mathbf{W}}_{1\times\mathsf{m}}\rangle$.

Fig. 5: Secure aggregation – Approach III (SepAgg [15]).

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 simulate the aggregation of random vectors $\vec{v}_i$ 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 *agg* computed on secret-shared and quantized inputs to the plain averaged aggregation $agg_{orig} = \sum_i^{\mathsf{n}} \vec{v}_i/\mathsf{n}$. Concretely, we measure

$$NMSE = \frac{\|agg_{orig} - agg\|_2^2}{\sum_i^{\mathsf{n}} \|\vec{v}_i\|_2^2/\mathsf{n}}, \qquad (7)$$

where *agg* is computed using various linear quantization schemes for (i) a regular dot product between converted bits and scales and (ii) with SepAgg. The code for the implementation of our simulation framework is available at https://encrypto.de/code/ScionFL. Our results shown in Fig. 6 are the average of 10 trials for each experiment with $\mathsf{q} = 3$ shares (representing a three-server dishonest majority setting using the masked evaluation technique). While for smaller 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 (e.g., random rotation and Kashin's representation) results in an unbiased aggregation is a significant theoretical challenge, left for future work.

Communication Costs: Tab. II provides the theoretical communication costs for our approaches when aggregating n quantized single-dimension vectors. Clearly, our Approach-III (cf. Fig. 5) is the most efficient, with the multiplication-related cost being completely independent of the number of clients n due to SepAgg [15]. The concrete communication costs are provided in §III-E.

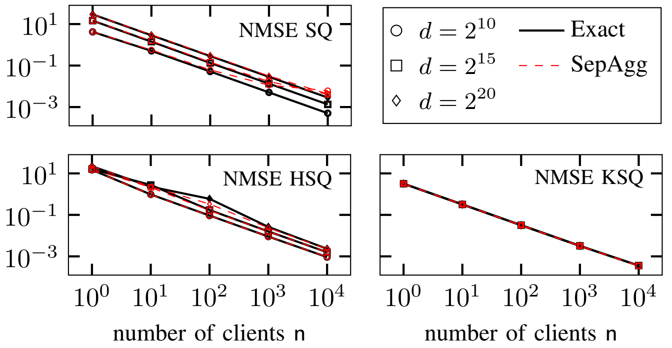


Fig. 6: NMSE comparison between exact and SepAgg-based aggregation for vanilla SQ, SQ using the randomized Hadamard transform (HSQ), and SQ using Kashin's representation (KSQ) for various vector dimensions d and number of clients n.

The communication costs are primarily determined by the cost of $\mathsf{BitA}^{\mathsf{pre}}$. In our masked evaluation technique (cf. §B-A), this relates to the conversion of a random secret-shared bit from Boolean sharing to its additive sharing form [42], [113]. Thus, we propose approximate variants for improving the cost of this operation.

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

TABLE II: Communication costs for aggregating quantized vectors with a single dimension for n clients. Protocols Π_{BitA} and Π_{Mult} are treated as black-boxes, and their costs are represented as BitA and Mult, respectively. The superscript pre in the costs denotes preprocessing and on denotes the online phase.

B. Approximate Bit Conversion in MPC

We reduce communication and computation costs with a novel *approximate* bit conversion method. Consider a bit b represented using two Boolean shares $\mathsf{b}_1, \mathsf{b}_2 \in \{0,1\}$, such that $\mathsf{b} = \mathsf{b}_1 \oplus \mathsf{b}_2$. Note that when embedding b_1 and b_2 in a larger field/ring[5], it holds that $\mathsf{b} = \mathsf{b}_1 + \mathsf{b}_2 - 2\mathsf{b}_1\mathsf{b}_2$. Similarly, for $\mathsf{b} = \mathsf{b}_1 \oplus \mathsf{b}_2 \oplus \mathsf{b}_3$, $\mathsf{b} = \mathsf{b}_1 + \mathsf{b}_2 + \mathsf{b}_3 - 2\mathsf{b}_1\mathsf{b}_2 - 2\mathsf{b}_1\mathsf{b}_3 - 2\mathsf{b}_2\mathsf{b}_3 + 4\mathsf{b}_1\mathsf{b}_2\mathsf{b}_3$ holds true. This concept generalizes to an arbitrary number of shares, denoted by q, as discussed below.

For $\mathsf{b} = \oplus_{i=1}^{\mathsf{q}} \mathsf{b}_i$, let $\mathcal{Q} = \{\mathsf{b}_i\}_{i\in[\mathsf{q}]}$ denote the set of all q shares of b, and $\tilde{\mathsf{b}}_i$ the arithmetic equivalent of the share b_i. Let $2^{\mathcal{Q}}$ be the powerset of $\mathcal{Q}$ and $\mathcal{Q}^{|c|}$ the set of all size-c subsets in $2^{\mathcal{Q}}$, that is, $2^{\mathcal{Q}} = \sum_{i=0}^{q} \mathcal{Q}^{|i|}$. The arithmetic equivalent of b, denoted by $\tilde{\mathsf{b}}$, is given as

$$\begin{aligned}\tilde{\mathsf{b}} &= \sum_{\{\mathsf{b}_e\}\in\mathcal{Q}^{|1|}} \tilde{\mathsf{b}}_e \; - \; 2 \; \cdot \sum_{\{\mathsf{b}_{e_1},\mathsf{b}_{e_2}\}\in\mathcal{Q}^{|2|}} \tilde{\mathsf{b}}_{e_1}\tilde{\mathsf{b}}_{e_2} \; + \; \ldots + (-2)^{\mathsf{q}-1} \; \cdot \prod_{\{\mathsf{b}_{e_1},\ldots,\mathsf{b}_{e_q}\}\in\mathcal{Q}^{|q|}} \tilde{\mathsf{b}}_e \\ &= \sum_{k=1}^{\mathsf{q}} (-2)^{k-1} \sum_{\{\mathsf{b}_{e_1},\ldots,\mathsf{b}_{e_k}\}\in\mathcal{Q}^{|k|}} \tilde{\mathsf{b}}_{e_1}\tilde{\mathsf{b}}_{e_2}\ldots\tilde{\mathsf{b}}_{e_k} \qquad (8)\end{aligned}$$

[4] We use this distribution for preliminary measurements as it was commonly observed in neural network gradients, e.g., [38].

[5] The bit (either 0 or 1) is treated as a ring element in $\mathbb{Z}_{2^\ell}$ in our protocols.

Note that the Eq. (8) can be viewed as sum of three terms: Sum ($\mathsf{term_s}$), Middle ($\mathsf{term_m}$), and Product ($\mathsf{term_p}$) as shown in Eq. (9) below. (Note that $\mathcal{Q}^{|q|} = \mathcal{Q}$).

$$\tilde{\mathsf{b}} = \underbrace{\sum_{\{\mathsf{b}_e\}\in\mathcal{Q}^{|1|}} \tilde{\mathsf{b}}_e}_{\text{Sum Term: } \mathsf{term_s}} + \underbrace{\sum_{k=2}^{\mathsf{q}-1}(-2)^{k-1}\sum_{\{\mathsf{b}_{e_1},\dots,\mathsf{b}_{e_k}\}\in\mathcal{Q}^{|k|}} \tilde{\mathsf{b}}_{e_1}\tilde{\mathsf{b}}_{e_2}\dots\tilde{\mathsf{b}}_{e_k}}_{\text{Middle Term: } \mathsf{term_m}} + \underbrace{(-2)^{\mathsf{q}-1}\prod_{\mathsf{b}_e\in\mathcal{Q}} \tilde{\mathsf{b}}_e}_{\text{Product Term: } \mathsf{term_p}} \tag{9}$$

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 a large number of shares is involved.

To approximate $\tilde{\mathsf{b}}$ in Eq. (9), we replace only term $\mathsf{term_m}$ with its expected value $\mathbb{E}[\mathsf{term_m}]$ such that the approximate value of $\tilde{\mathsf{b}}$, denoted by $\hat{\mathsf{b}}$, retains $\mathbb{E}[\hat{\mathsf{b}}] = \mathsf{b}$. The expectation of $\mathsf{term_s}$ and $\mathsf{term_p}$ in Eq. (9) is first calculated, and $\mathbb{E}[\mathsf{term_m}]$ is inferred using the fact that $\mathbb{E}[\hat{\mathsf{b}}] = \mathsf{b}$. This analysis is summarised in Lem. III.1 and the proof is provided in §C-B.

Lemma III.1 (Expected Values)**.** *Given a bit* $\mathsf{b} = \oplus_{i=1}^{\mathsf{q}}\mathsf{b}_i$ *and* $b = \mathsf{term_s} + \mathsf{term_m} + \mathsf{term_p}$ *with*

$$\mathsf{term_s} = \sum_{\{\mathsf{b}_e\}\in\mathcal{Q}^{|1|}} \tilde{\mathsf{b}}_e, \quad \mathsf{term_m} = \sum_{k=2}^{\mathsf{q}-1}(-2)^{k-1}\sum_{\{\mathsf{b}_{e_1},\dots,\mathsf{b}_{e_k}\}\in\mathcal{Q}^{|k|}} \tilde{\mathsf{b}}_{e_1}\tilde{\mathsf{b}}_{e_2}\dots\tilde{\mathsf{b}}_{e_k},$$

$$\mathsf{term_p} = (-2)^{\mathsf{q}-1}\prod_{\mathsf{b}_e\in\mathcal{Q}} \tilde{\mathsf{b}}_e,$$

we have $\mathbb{E}[\mathsf{term_s} \mid \mathsf{b}] = \mathsf{q}/2$*,* $\mathbb{E}[\mathsf{term_m} \mid \mathsf{b}] = (\mathsf{q}\text{-}1) \bmod 2 - \mathsf{q}/2$*, and* $\mathbb{E}[\mathsf{term_p} \mid \mathsf{b}] = \mathsf{b} - (\mathsf{q}\text{-}1) \bmod 2$.

Our Approximation. We define the approximate arithmetic equivalent of b, denoted by $\hat{\mathsf{b}}$, as follows:

$$\hat{\mathsf{b}} = \underbrace{\sum_{\mathsf{b}_e\in\mathcal{Q}} \tilde{\mathsf{b}}_e}_{\mathsf{term_s}} + \underbrace{\left((\mathsf{q}\text{-}1) \bmod 2 - \frac{\mathsf{q}}{2}\right)}_{\mathsf{term}^{\mathsf{a}}_{\mathsf{m}}} + \underbrace{(-2)^{\mathsf{q}-1}\prod_{\mathsf{b}_e\in\mathcal{Q}} \tilde{\mathsf{b}}_e}_{\mathsf{term_p}} \tag{10}$$

While $\mathsf{term_s}$ 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 $\mathsf{term_p}$ is required to keep the expected values for $\mathsf{b} = 0$ and $\mathsf{b} = 1$ different. This is evident from Lem. III.1 where $\mathbb{E}[\mathsf{term_p} \mid \mathsf{b}]$ 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 $\mathbb{E}[\mathsf{b} = 0] = \mathbb{E}[\mathsf{b} = 1]$. The intuition is that only such a term can differentiate between $\mathsf{b} = 0$ and $\mathsf{b} = 1$, while all other terms will be symmetrically distributed. For instance, consider $\mathsf{q} = 3$ and let $\tilde{\mathsf{b}} = c_1\tilde{\mathsf{b}}_1 + c_2\tilde{\mathsf{b}}_2 + c_3\tilde{\mathsf{b}}_3 + c_4\tilde{\mathsf{b}}_1\tilde{\mathsf{b}}_2 + c_5\tilde{\mathsf{b}}_2\tilde{\mathsf{b}}_3 + c_6\tilde{\mathsf{b}}_1\tilde{\mathsf{b}}_3 + c_7$ for some random combiners $c_i \in \mathbb{Z}_{2^\ell}$ and $i \in [7]$. Using the truth table T_{b} given in Tab. VII, it is easy to verify that

$$\mathbb{E}[\tilde{\mathsf{b}} = 0] = \mathbb{E}[\tilde{\mathsf{b}} = 1] = \frac{1}{4}\cdot(2c_1 + 2c_2 + 2c_3 + c_4 + c_5 + c_6 + 4c_7) \tag{11}$$

This argument can be generalized to any value of q.

Claim III.2. *The approximate arithmetic equivalent* $\hat{\mathsf{b}}$ *in Eq.* (10) *preserves the expectation of the exact bit* b *in Eq.* (8)*, i.e.,* $\mathbb{E}[\hat{\mathsf{b}} = 0] = 0$ *and* $\mathbb{E}[\hat{\mathsf{b}} = 1] = 1$.

Proof. The proof is straightforward as we replace the middle term ($\mathsf{term_m}$) in Eq. (8) with its expected value $\mathsf{term}^{\mathsf{a}}_{\mathsf{m}}$. □

We provide more details regarding the efficiency of the approximation in §C-B.

C. Secure Bit Aggregation with Global Scales

Here, we consider secure bit aggregation in the context of “global scales”, as discussed in §II-B. In this case, all the clients use the same set of scales for quantization, denoted by s^{min}_G and s^{max}_G. Therefore, it is sufficient to compute

$$\vec{\mathbf{X}}_{1\times\mathsf{m}} = \mathsf{s}^{min}_G \;\oplus\; \mathsf{Agg\text{-}R}\left(\vec{\mathbf{B}}_{\mathsf{n}\times\mathsf{m}}\right) \circ (\mathsf{s}^{max}_G - \mathsf{s}^{min}_G) \tag{12}$$

as the aggregation result. Interestingly, when $\mathsf{s}^{min}_G = 0$ and $\mathsf{s}^{max}_G = 1$, this can also be viewed as an instance of privacy-preserving aggregate statistics computation, as demonstrated in the works of Prio [39] and Prio+ [2].

As shown in Eq. (12), the computation becomes simpler in the case of global scales since all clients utilize the same set of public scales, denoted by s^{min} and s^{max}, to compute their quantized vector that corresponds to the rows of $\vec{\mathbf{B}}$. Hence, we just need to compute the column-wise aggregate of the $\vec{\mathbf{B}}$ matrix and use protocol $\Pi^{\mathsf{sum}}_{\mathsf{BitA}}$ (Fig. 16 in §B-A) to do so. The resulting protocol $\Pi^{\mathsf{Global}}_{\mathsf{SecAgg}}$ appears in Fig. 7.

Protocol $\Pi^{\mathsf{Global}}_{\mathsf{SecAgg}}(\langle\vec{\mathbf{B}}_{\mathsf{n}\times\mathsf{m}}\rangle^{\mathbf{B}}, \mathsf{s}^{min}, \mathsf{s}^{max})$

1. Compute $\langle\vec{\mathbf{W}}^j\rangle = \Pi^{\mathsf{sum}}_{\mathsf{BitA}}(\langle\vec{\mathbf{B}}^j\rangle^{\mathbf{B}})$, for each $j \in [\mathsf{m}]$.
2. Locally compute $\langle\vec{\mathbf{Y}}_{1\times\mathsf{m}}\rangle = \mathsf{s}^{min} \oplus \left(\langle\vec{\mathbf{W}}_{1\times\mathsf{m}}\rangle\cdot(\mathsf{s}^{max} - \mathsf{s}^{min})\right)$.

Fig. 7: Secure aggregation – Global Scales.

D. Accuracy Evaluation

In §III-B, 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 run a simulation similar to the one described in §III-A. Here, we compare the NMSE computed as in Eq. (7) for an aggregation *agg* when using various linear quantization schemes with global scales (i) with an exact bit-to-arithmetic conversion and (ii) with our approximation enabled. The implementation is available at https://encrypto.de/code/ScionFL. Our results in Fig. 8 are the average of 10 trials for each experiment with $\mathsf{q} = 3$ shares. Consistently, we observe that 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. 9, we 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. Furthermore, as shown in §III-F, the error is still so small that the impact on the accuracy in common FL settings is negligible.

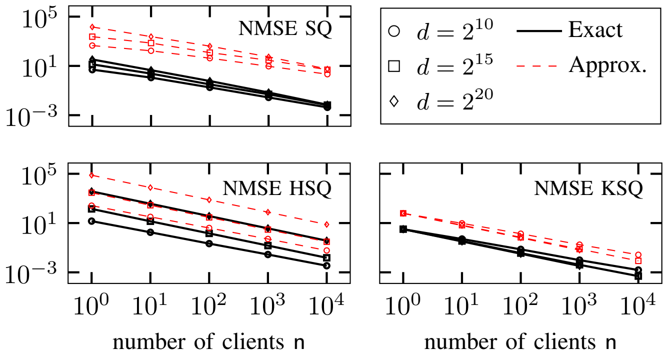


Fig. 8: NMSE comparison between exact and approximation-based aggregation for vanilla SQ, SQ using the randomized Hadamard transform (HSQ), and SQ using Kashin's representation (KSQ) for global scales with q = 3 shares and various vector dimensions d and number of clients n.

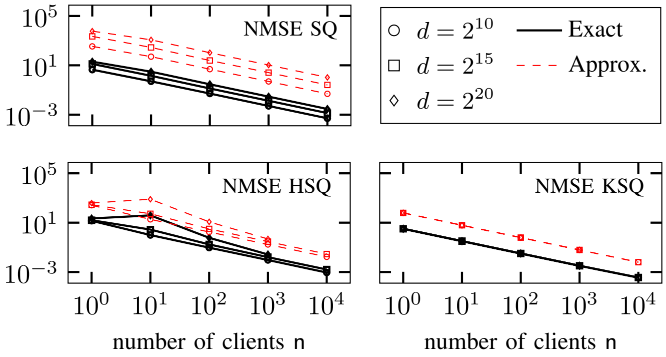


Fig. 9: NMSE comparison between exact and approximation-based aggregation for SQ, Hadamard SQ (HSQ), and Kashin SQ (KSQ) for local scales with q = 3 shares, various vector dimensions d, and number of clients n.

E. Detailed Communication Costs

Here, we provide more insights into the concrete communication costs for our secure aggregation protocols in §III-A.

In Tab. III we provide the detailed communication costs for the secure aggregation approaches discussed in §III-A when training the LeNet architecture for image classification on the MNIST data set [83] using 1-bit SQ with Kashin's representation [31]. We instantiate the OT instances required in the preprocessing phase, as discussed in §B-A, with silent OT [40], following Prio+ [2]. Here, we can observe the significant impact of including SepAgg [15] in practice with performance improvements between Approach-II and Approach-III of up to $16.6\times$ in the offline phase.

In Tab. IV, we compare the aggregation of bits (i.e., when not considering quantized inputs that require scale multiplication and hence without SepAgg [15] being applicable) to Prio+ [2]. For a fair comparison, we translate the approach in Prio+ [2] to our three party dishonest-majority setting. As we can see, even for exact bit-to-arithmetic conversion, we improve over Prio+ by factor $2.4\times$ for n $= 10^5$. When applying our approximate bit-to-arithmetic conversion (cf. §III-B), this improvement increases to a factor of $4\times$.

n	Method	Exact		Approx.	
		Offline	Online	Offline	Online
20	Approach-I	644.50	1.70	620.27	1.70
	Approach-II	644.50	0.59	620.27	0.59
	Approach-III	89.77	0.59	65.54	0.59
100	Approach-I	3222.51	6.12	3101.36	6.12
	Approach-II	3222.51	0.59	3101.36	0.59
	Approach-III	332.08	0.59	210.93	0.59
500	Approach-I	16112.56	28.24	15506.80	28.24
	Approach-II	16112.56	0.59	15506.80	0.59
	Approach-III	1543.62	0.59	937.85	0.59

TABLE III: 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 (KSQ). We compare Approach-I (cf. Fig. 3 in §III-A), Approach-II (cf. Fig. 4 in §III-A), and Approach-III (cf. Fig. 5 in §III-A). Additionally, we distinguish between using an exact bit-to-arithmetic conversion and our approximation (cf. §III-B).

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 IV: Total communication in MiB of Approach-III (cf. Fig. 5 in §III-A) compared to Prio+ [2] to calculate the sum of bits for different numbers of clients n and dimension m $= 1000$. For Approach-III, we distinguish between using an exact bit-to-arithmetic conversion as in Prio+ [2] and our approximation (cf. §III-B).

F. Performance Evaluation

We implemented an extensive end-to-end FL evaluation and MPC simulation framework. We describe our implementation, the parameters for our accuracy evaluation, and present the results.

Implementation: Our implementation is written in Python based on PyTorch. It supports multi-GPU acceleration, also for our MPC simulation. We used a subset of this framework for measuring the accuracy of SepAgg (cf. §III-A) and our approximate bit conversion (cf. §III-D), and we will describe extensions in §IV-B to incorporate evaluations of poisoning attacks and defenses.

Our framework provides a command-line interface 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 training. 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 §III-A will be executed locally. Note that our goal is not to assess the run-time performance of the MPC protocol but rather precisely measure the impact on 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 (i) LeNet

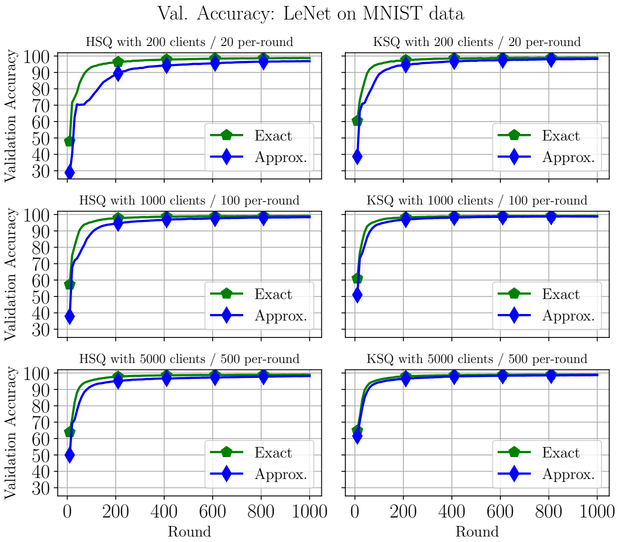


Fig. 10: Validation accuracy for training LeNet on the MNIST data set for $\mathsf{n} \in \{200, 1000, 5000\}$ clients when selecting 10% of the clients at random per round (n) for SQ with Hadamard (HSQ, left) and with Kashin's representation (KSQ, right); "Exact" denotes the insecure baseline, "Approx" the simulation of our MPC-based approximate secure aggregation including SepAgg (cf. Fig. 5).

on MNIST [83] for 1000 rounds and (ii) ResNet9 on CIFAR-10 [80] 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 $\mathsf{n} = 40$ 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. §III-D), 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. 5), we choose a three-server dishonest majority setting.

Results: The results for the MNIST/LeNet training are given in Fig. 10. 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 observations apply to CIFAR10/ResNet9 in Fig. 11. However, here the difference between the exact and approximate version for KSQ is higher with 3.14% in the final round. 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. 6 and 8). 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 model architectures due to hardware limitations.

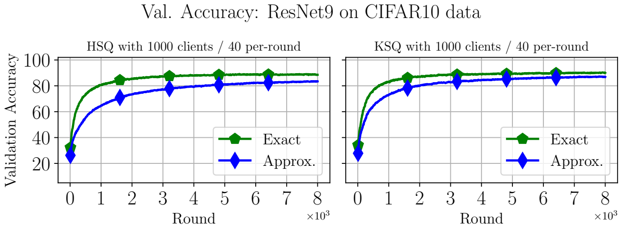


Fig. 11: Validation accuracy for training ResNet9 on the CIFAR10 data set for $N = 1000$ clients with random $\mathsf{n} = 40$ selected per round for quantization techniques and protocols as in Fig. 10.

Additionally, one may use a *hybrid approach*, where training uses the approximate version for initial rounds until a baseline accuracy is reached, whereas secure exact training (potentially including only the SepAgg [15] approximation but not our approximate bit-to-arithmetic conversion) is used for fine tuning up to the desired target accuracy.

In Tab. V, 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 ≈ 15×. For the online communication, we can see a wide range of improvement factors from 2.9× to 48× for MNIST with $\mathsf{n} = 500$. This highlights the positive impact when utilizing the SepAgg approach for aggregating an increasingly large number of rows 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 §B-C, 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.

			Naive Exact (cf. Fig. 3)		Our Approx. (cf. Fig. 5)	
Benchmark	n	Method	Offline	Online	Offline	Online
MNIST/ LeNet	20	HSQ	572.89	1.51	58.26	0.52
		KSQ	644.50	1.70	65.54	0.59
MNIST/ LeNet	100	HSQ	2864.46	5.44	187.49	0.52
		KSQ	3222.51	6.12	210.93	0.59
MNIST/ LeNet	500	HSQ	14322.28	25.10	833.64	0.52
		KSQ	16112.56	28.24	937.85	0.59
CIFAR10/ ResNet9	40	HSQ	87079.45	189.27	6883.13	39.85
		KSQ	100828.84	219.15	7969.94	46.14

TABLE V: Inter-server communication per round for our benchmarks for different numbers of clients n in MiB.

IV. Defending Untargeted Poisoning Attacks

Our defense called ScionFL-Aura is designed to mitigate untargeted poisoning attacks in the context of secure quantized aggregation. These attacks pose a significant threat to the deployment of FL for two reasons: (i) Untargeted attacks are particularly difficult to detect because, ignorant of the attack, service providers are unaware that they could have achieved a greater accuracy. (ii) Even a minor drop in accuracy can cause enormous (competitive) damage [119].

Most proposed untargeted poisoning attacks on FL use the (unrealistic) assumption that the adversary $\mathcal{A}$ is aware of either the aggregation rule [52] or all benign updates [11]. However, the *Min-Max* attack proposed by [118] 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; the attacker can then 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 [118, §IV] for more specifics on the attack.

In addition to removing assumptions about the adversary's knowledge, [118] empirically shows that the *Min-Max* attack outperforms the former state-of-the-art poisoning attack [11] for almost all tested datasets. However, since all benchmarks in [11], [118] were performed on FL schemes without quantization, the impact of the *Min-Max* attack on quantized FL schemes is unclear. Hence, we first test the attack's effectiveness in our framework using the open-sourced code[6] as baseline. As we discuss in §IV-B, we observe that the attacks are effective even in the context of quantization.

A. 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 works to propose defense mechanisms [118], [102], [110], however, those cannot be combined trivially with ScionFL without having to de-quantize all updates and running expensive secure computation machinery.

We now outline the general design of ScionFL-Aura and show its effectiveness against the *Min-Max* attack [118]. In §IV-A, we describe how to efficiently instantiate it in an MPC-friendly manner to reduce communication overhead.

Approach: ScionFL-Aura uses a hybrid approach, combining ideas from existing FL defenses based on the L_2-norm [10], [102], [124] and cosine similarity [102], [32]. Several works like [102] compute these metrics for each client pair, resulting in expensive computation. In contrast, we aggregate all updates, including the poisoned ones, to produce the vector $\vec{X}^{\mathsf{agg}}$, which we then utilize as the reference. At a high level, L_2-norm based scaling 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:

Algorithm 1 Our Defense: ScionFL-Aura

```
 1: procedure ScionFL-Aura($(\{\vec{\sigma}_{X_i}, \mathsf{s}_{X_i}^{min}, \mathsf{s}_{X_i}^{max}\}_{i\in[\mathsf{n}]})$)
    // Gradient Aggregation including poisoned ones.
 2:     $\vec{X}^{\mathsf{agg}} \leftarrow$ Aggregate$(\{\vec{\sigma}_{X_i}, \mathsf{s}_{X_i}^{min}, \mathsf{s}_{X_i}^{max}\}_{i\in\mathsf{n}})$
    // $\mathsf{L}_2$-norm Computation
 3:     $\mathsf{L}_2^{\mathsf{avg}} \leftarrow 0$
 4:     for $k \leftarrow 1$ to $\mathsf{n}$ do
 5:         $\mathsf{L}_2^k \leftarrow$ L2-NormQ$(\vec{\sigma}_{X_k}, \mathsf{s}_{X_k}^{min}, \mathsf{s}_{X_k}^{max})$
 6:         $\mathsf{L}_2^{\mathsf{avg}} \leftarrow \mathsf{L}_2^{\mathsf{avg}} + \mathsf{L}_2^k$
 7:     end for
    // $\mathsf{L}_2$-norm based Scaling
 8:     $\mathsf{L}_2^{\mathsf{avg}} \leftarrow \mathsf{L}_2^{\mathsf{avg}}/\mathsf{n}$ // Average of $\mathsf{L}_2$-norms
 9:     for $k \leftarrow 1$ to $\mathsf{n}$ do
10:         if $\mathsf{L}_2^k > \mu_{\mathsf{th}} \cdot \mathsf{L}_2^{\mathsf{avg}}$ then
11:             $\mathsf{s}_{X_k}^{min} \leftarrow \mathsf{s}_{X_k}^{min} \cdot (\mu_{\mathsf{th}} \cdot \mathsf{L}_2^{\mathsf{avg}})/\mathsf{L}_2^k$
12:             $\mathsf{s}_{X_k}^{max} \leftarrow \mathsf{s}_{X_k}^{max} \cdot (\mu_{\mathsf{th}} \cdot \mathsf{L}_2^{\mathsf{avg}})/\mathsf{L}_2^k$
13:         end if
14:     end for
    // Cosine-distance based Filtering
15:     for $k \leftarrow 1$ to $\mathsf{n}$ do
16:         $\theta^k \leftarrow$ Cosine$((\vec{\sigma}_{X_k}, \mathsf{s}_{X_k}^{min}, \mathsf{s}_{X_k}^{max}), \vec{X}^{\mathsf{agg}})$
17:     end for
18:     $\mathcal{X} \leftarrow$ Top-K$(\vec{\theta}, \psi)$ // Returns $k$ for which $\theta^k > \psi$
    // Aggregation of filtered updates
19:     $\vec{X}^{\mathsf{aggd}} \leftarrow$ Aggregate$(\{\vec{\sigma}_{X_i}, \mathsf{s}_{X_i}^{min}, \mathsf{s}_{X_i}^{max}\}_{i\in[\mathsf{n}], i\notin\mathcal{X}})$
20: end procedure
```

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 μ_{th} denote the threshold and $\mathsf{L}_2^{\mathsf{avg}}$ denote the average of the L_2-norms across all clients. If $\mathsf{L}_2^X > \mu_{\mathsf{th}} \cdot \mathsf{L}_2^{\mathsf{avg}}$ for a gradient vector $\vec{X}$, the vector is scaled[7] by a factor of $\mu_{\mathsf{th}} \cdot \mathsf{L}_2^{\mathsf{avg}}/\mathsf{L}_2^X$. This ensures that no gradient has an L_2-norm greater than $\mu_{\mathsf{th}} \cdot \mathsf{L}_2^{\mathsf{avg}}$.

2) *Cosine-distance-based Filtering.* This step computes the cosine distance for each gradient from the reference vector $\vec{X}^{\mathsf{agg}}$. After that, another aggregation is carried out on the updated vectors, excluding the top-ψ vectors with the highest cosine distances using a secure Top-K algorithm, which involves sorting and selecting the first K items. Here, ψ is either a a known bound (i.e., defined in advance by the service provider) or an accepted percentile determined based on an assumed attacker ratio following a normal distribution.

Alg. 1 provides the formal details of ScionFL-Aura, 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.

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, for which we discuss viable alternatives below.

1) *(Line 5 in Alg. 1).* The computation of L_2-norm within L2-NormQ (cf. Alg. 3 in §C-C) involves calculating the square

[6] https://github.com/vrt1shjwlkr/NDSS21-Model-Poisoning

[7] Scaling a quantized vector requires simply scaling the scales (cf. §II-D).

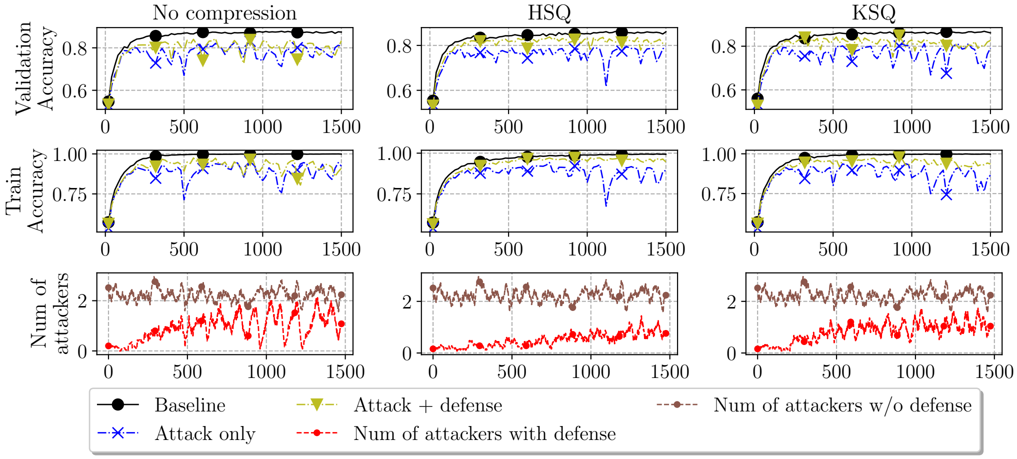


Fig. 12: Effect of *Min-Max* attack [118] on training ResNet9 with CIFAR10 for 1500 aggregation rounds with and without our defense ScionFL-Aura assuming 20% of $N = 50$ clients are corrupted. Note that the number of attackers included in the global update varies even without defense due to random client selection.

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 [33], [89].

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 $\vec{X}^{\mathsf{agg}}$ requires computing the L_2-norm of $\vec{X}^{\mathsf{agg}}$ and dividing by it, as shown in COSINE (cf. Alg. 4 in §C-C). However, the cosine distances are only used to filter out the top-ψ 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 $\vec{X}^{\mathsf{agg}}$ when computing the cosine distance for our purpose.

In addition to the aforementioned optimizations, most of the values computed as part of the $\vec{X}^{\mathsf{agg}}$ computation in the 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. §C-C provides details on the sub-protocols utilized in our defense algorithm given in Alg. 1.

B. Effectiveness Evaluation

To analyze the effectiveness of ScionFL-Aura, we test it against the *Min-Max* attack [118].

Setup: Training involves $N = 50$ clients of which 20% (as in [118]) are corrupted. Per training iteration, a random subset of $\mathsf{n} = 10$ 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 of $\eta_{\mathsf{C}} = 0.1$. The defense threshold μ_{th} is set to 3 and the momentum is 0.9.[8]

Experimental Results: Our results when training ResNet9 on CIFAR10 (i) without an attack, (ii) under attack without defense, and (iii) under attack with ScionFL-Aura in place are given in Fig. 12. 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). We also provide similar results for training VGG11 in §C-C. As shown in Fig. 12, 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 [118], where the authors report an accuracy degradation between 10.1% and 42.1% for CIFAR10, depending on the model architecture and the aggregation scheme. Furthermore, our experiments show that quantization does not significantly change the impact of the attack. When 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 attacker's 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.

ACKNOWLEDGMENTS

This project received funding from the ERC under the EU's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the DFG within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by the Private AI Collaborative Research Institute, funded by Intel, Avast, and VMware.

[8] [119] 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 [118].

References

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

[2] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou, "Prio+: Privacy Preserving Aggregate Statistics via Boolean Shares," in *SCN*, 2022.

[3] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *EMNLP*, 2017.

[4] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The Convergence of Sparsified Gradient Methods," in *NeurIPS*, 2018.

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

[6] Apple and Google, "Exposure Notification Privacy-preserving Analytics (ENPA) White Paper," https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf.

[7] 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.

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

[9] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *ACM CCS*, 2013.

[10] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How To Backdoor Federated Learning," in *AISTATS*, 2020.

[11] G. Baruch, M. Baruch, and Y. Goldberg, "A Little Is Enough: Circumventing Defenses For Distributed Learning," in *NeurIPS*, 2019.

[12] 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.

[13] 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.

[14] C. Baum, I. Damgård, T. Toft, and R. W. Zakarias, "Better Preprocessing for Secure Multiparty Computation," in *ACNS*, 2016.

[15] C. Beguier, M. Andreux, and E. W. Tramel, "Efficient Sparse Secure Aggregation for Federated Learning," 2020, https://arxiv.org/abs/2007.14861.

[16] 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.

[17] A. Ben-Efraim, M. Nielsen, and E. Omri, "Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing," in *ACNS*, 2019.

[18] J. Bernstein, Y. Wang, K. Azizzadenesheli, and A. Anandkumar, "SIGNSGD: Compressed Optimisation for Non-Convex Problems," in *ICML*, 2018.

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

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

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

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

[23] 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.

[24] ——, "All You Need Is Matplotlib," http://www.cleverhans.io/2022/04/17/fl-privacy.html, 2022.

[25] ——, "Is Federated Learning a Practical PET Yet?" *CoRR*, 2023.

[26] 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.

[27] 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.

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

[29] A. Brüggemann, O. Schick, T. Schneider, A. Suresh, and H. Yalame, "Don't Eject the Impostor: Fast Three-Party Computation With a Known Cheater," in *IEEE S&P*, 2024.

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

[31] 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.

[32] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," in *NDSS*, 2021.

[33] O. Catrina and A. Saxena, "Secure Computation with Fixed-Point Numbers," in *FC*, 2010.

[34] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction," in *ACM Conference on Cloud Computing Security Workshop, CCSW@CCS*, 2019.

[35] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning," in *NDSS*, 2020.

[36] 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.

[37] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *ASIACRYPT*. Springer, 2017.

[38] 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.

[39] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics," in *USENIX NSDI*, 2017.

[40] G. Couteau, P. Rindal, and S. Raghuraman, "Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes," in *CRYPTO*, 2021.

[41] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "SPDZ2^k: Efficient MPC mod 2^k for Dishonest Majority," in *CRYPTO*, 2018.

[42] 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.

[43] 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.

[44] I. Damgård, C. Orlandi, and M. Simkin, "Yet Another Compiler for Active Security or: Efficient MPC Over Arbitrary Rings," in *CRYPTO*, 2018.

[45] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO*, 2012.

[46] P. Davies, V. Gurunanthan, N. Moshrefi, S. Ashkboos, and D. Alistarh, "New bounds for distributed mean estimation and variance reduction," in *ICLR*, 2021.

[47] D. Demmler, T. Schneider, and M. Zohner, "ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation," in *NDSS*, 2015.

[48] 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.

[49] T. Elahi, G. Danezis, and I. Goldberg, "PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks," in *ACM CCS*, 2014.

[50] 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.

[51] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl, "Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits," in *CRYPTO*, 2020.

[52] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local Model Poisoning Attacks to Byzantine-Robust Federated Learning," in *USENIX Security*, 2020.

[53] 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.

[54] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. D. Nguyen, P. Rieger, A. Sadeghi, T. Schneider, H. Yalame, and S. Zeitouni, "SAFELearn: Secure Aggregation for private FEderated Learning," in *IEEE S&P Workshops*, 2021.

[55] L. H. Fowl, J. Geiping, W. Czaja, M. Goldblum, and T. Goldstein, "Robbing the Fed: Directly Obtaining Private Data in Federated Learning with Modified Models," in *ICLR*, 2022.

[56] 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.

[57] 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.

[58] T. Gehlhar, F. Marx, T. Schneider, T. Wehrle, A. Suresh, and H. Yalame, "SafeFL: MPC-friendly framework for Private and Robust Federated Learning," in *IEEE S&P Workshops*, 2023.

[59] 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.

[60] S. D. Gordon, S. Ranellucci, and X. Wang, "Secure Computation with Low Communication from Cross-Checking," in *ASIACRYPT*, 2018.

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

[62] A. Hegde, H. Möllering, T. Schneider, and H. Yalame, "SoK: Efficient Privacy-preserving Clustering," *PETS*, 2021.

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

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

[65] 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.

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

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

[68] 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.

[69] 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.

[70] 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.

[71] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer," in *ACM SIGSAC*, 2016.

[72] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ Great Again," in *EUROCRYPT*, 2018.

[73] M. Keller, P. Scholl, and N. P. Smart, "An architecture for practical actively secure MPC with dishonest majority," in *ACM CCS*, 2013.

[74] 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.

[75] V. Kolesnikov and T. Schneider, "Improved Garbled Circuit: Free XOR Gates and Applications," in *ICALP*, 2008.

[76] 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.

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

[78] N. Koti, S. Patil, A. Patra, and A. Suresh, "MPClan: Protocol suite for privacy-conscious computations," *J. Cryptol.*, 2023.

[79] N. Koti, A. Patra, R. Rachuri, and A. Suresh, "Tetrad: Actively Secure 4PC for Secure Training and Inference," in *NDSS*, 2022.

[80] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[81] K. Kursawe, G. Danezis, and M. Kohlweiss, "Privacy-Friendly Aggregation for the Smart-Grid," in *PETS*, 2011.

[82] 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.

[83] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, 1998.

[84] 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.

[85] 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.

[86] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," in *MLSys*, 2020.

[87] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai, "Efficient Constant Round Multi-party Computation Combining BMR and SPDZ," in *CRYPTO*, 2015.

[88] Y. Lyubarskii and R. Vershynin, "Uncertainty principles and vector quantization," *IEEE Trans. Inf. Theory*, 2010.

[89] E. Makri, D. Rotaru, F. Vercauteren, and S. Wagh, "Rabbit: Efficient Comparison for Secure Multi-Party Computation," in *FC*, 2021.

[90] M. Mansouri, M. Önen, W. Ben Jaballah, and M. Conti, "SoK: Secure aggregation based on cryptographic schemes for federated Learning," in *PETS*, 2023.

[91] T. Marchand, R. Loeb, U. Marteau-Ferey, J. O. du Terrail, and A. Pignet, "SRATTA: sample re-attribution attack of secure aggregation in federated learning," in *ICML*, 2023.

[92] F. Marx, T. Schneider, A. Suresh, T. Wehrle, C. Weinert, and H. Yalame, "Hyfl: A hybrid approach for private federated learning," 2023, https://arxiv.org/abs/2302.09904.

[93] S. Mazloom, P. H. Le, S. Ranellucci, and S. D. Gordon, "Secure parallel computation on national scale volumes of data," in *USENIX Security*, 2020.

[94] 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.

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

[96] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *USENIX Security*, 2020.

[97] P. Mohassel and P. Rindal, "ABY^3: A Mixed Protocol Framework for Machine Learning," in *ACM CCS*, 2018.

[98] P. Mohassel and Y. Zhang, "SecureML: A System for Scalable Privacy-Preserving Machine Learning," in *IEEE S&P*, 2017.

[99] 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.

[100] J. Münch, T. Schneider, and H. Yalame, "VASA: Vector AES Instructions for Security Applications," in *ACM ACSAC*, 2021.

[101] 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.

[102] T. D. Nguyen, P. Rieger, H. Chen, H. Yalame, H. Möllering, 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.

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

[104] D. Pasquini, D. Francati, and G. Ateniese, "Eluding Secure Aggregation in Federated Learning via Model Inconsistency," in *CCS*, 2022.

[105] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation," in *USENIX Security*, 2021.

[106] ——, "SynCirc: Efficient Synthesis of Depth-Optimized Circuits for Secure Computation," in *IEEE HOST*, 2021.

[107] A. Patra and A. Suresh, "BLAZE: Blazing Fast Privacy-Preserving Machine Learning," in *NDSS*, 2020.

[108] R. A. Popa, A. J. Blumberg, H. Balakrishnan, and F. H. Li, "Privacy and accountability for location-based aggregate statistics," in *ACM CCS*, 2011.

[109] A. Pyrgelis, C. Troncoso, and E. D. Cristofaro, "Knock Knock, Who's There? Membership Inference on Aggregate Location Data," in *NDSS*, 2018.

[110] M. Rathee, C. Shen, S. Wagh, and R. A. Popa, "ELSA: Secure Aggregation for Federated Learning with Malicious Actors," in *IEEE S&P*, 2023.

[111] 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.

[112] P. Richtárik, I. Sokolov, and I. Fatkhullin, "EF21: A New, Simpler, Theoretically Better, and Practically Faster Error Feedback," in *NeurIPS*, 2021.

[113] D. Rotaru and T. Wood, "MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security," in *INDOCRYPT*, 2019.

[114] 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.

[115] 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.

[116] 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.

[117] 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.

[118] V. Shejwalkar and A. Houmansadr, "Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning," in *NDSS*, 2021.

[119] 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.

[120] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership Inference Attacks Against Machine Learning Models," in *IEEE S&P*, 2017.

[121] J. So, R. E. Ali, B. Guler, J. Jiao, and S. Avestimehr, "Securing Secure Aggregation: Mitigating Multi-Round Privacy Leakage in Federated Learning," *AAAI*, 2021.

[122] J. So, B. Güler, and A. S. Avestimehr, "Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning," *IEEE J. Sel. Areas Inf. Theory*, 2021.

[123] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with Memory," in *NeurIPS*, 2018.

[124] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, "Can you really backdoor federated learning?" in *NeurIPS FL Workshop*, 2019.

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

[126] A. T. Suresh, F. X. Yu, S. Kumar, and H. B. McMahan, "Distributed Mean Estimation with Limited Communication," in *ICML*, 2017.

[127] 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.

[128] 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.

[129] S. Vargaftik, R. Ben-Basat, A. Portnoy, G. Mendelson, Y. Ben-Itzhak, and M. Mitzenmacher, "DRIVE: One-bit Distributed Mean Estimation," in *NeurIPS*, 2021.

[130] T. Verma and S. Singanamalla, "Improving DNS Privacy with Oblivious DoH in 1.1.1.1," https://blog.cloudflare.com/oblivious-dns/l, 2020.

[131] 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.

[132] 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.

[133] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *INFOCOM*, 2019.

[134] 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.

[135] Y. Wen, J. Geiping, L. Fowl, M. Goldblum, and T. Goldstein, "Fishing for User Data in Large-Batch Federated Learning via Gradient Magnification," in *ICML*, 2022.

[136] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is Feature Selection Secure against Training Data Poisoning?" in *ICML*, 2015.

[137] 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.

[138] A. C.-C. Yao, "Protocols for Secure Computations (Extended Abstract)," in *FOCS*, 1982.

[139] D. Yin, Y. Chen, K. Ramchandran, and P. L. Bartlett, "Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates," in *ICML*, 2018.

[140] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning," in *USENIX ATC*, 2020.

[141] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong, "FLDetector: Defending Federated Learning Against Model Poisoning Attacks via Detecting Malicious Clients," in *KDD*, 2022.

[142] 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.

Appendix A
Related Work & Background Information

A. Stochastic Quantization

This section provides additional details regarding stochastic quantization schemes discussed in §II-B.

Preprocessing 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 [126]. 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[9] of $O(\frac{\log \mathsf{m}}{\mathsf{n}})$ using $O(\mathsf{m})$ bits, which asymptotically improves over the $O(\frac{\mathsf{m}}{\mathsf{n}})$ NMSE bound of vanilla SQ. The computational complexity, on the other hand, is increased from $O(\mathsf{m})$ to $O(\mathsf{m} \log \mathsf{m})$ when utilizing the randomized Hadamard transform for rotations.

Preprocessing via Kashin's Representation: The rotation approach was recently improved using Kashin's representation [31], [88], [115]. Roughly speaking, it allows representing an m-dimensional vector using a slightly larger vector with $\lambda \cdot \mathsf{m}$ smaller coefficients ($\lambda > 1$). It can be shown that applying SQ to the Kashin coefficients allows for an optimal NMSE of $O(\frac{1}{\mathsf{n}})$ using $O(\lambda \cdot \mathsf{m})$ bits. Compared with [126], Kashin's representation yields a lower NMSE bound by a factor of $\log \mathsf{m}$ at the cost of increasing the computational complexity by the same factor [13], [31].

[9] The normalized MSE is the mean's estimate MSE normalized by the mean clients' gradient squared norms

B. 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 like [53], [123], [3], [76] consider sparsifying the gradients. Quantization can also be applied to these sparsified gradients as it reduces the number of bits used per entry, while sparsification reduces the number of entries.

Client-side Memory-based Techniques: Some compression techniques, including Top-k [123] and sketching [64], rely on client-side memory and error-feedback [117], [4], [112], [19] 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., [126], [128], [4]). While such techniques are appealing in their bandwidth-to-accuracy trade-offs, it is unclear how to allow for an efficient secure aggregation as gradients must be decoded before being averaged. Also, such techniques usually incur a higher computational overhead at the clients than fixed-length representations. An additional review of current state-of-the-art gradient compression techniques and some open challenges can be found in [69], [76], [131].

C. Secure Multi-party Computation

The field of secure multi-party computation (MPC) started with the seminal work of Yao [138] 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., [75], [9], [27], [51], [100]. Also, tailored efficient optimizations for varying number of computation parties have been explored, e.g., [47], [105], [34], [35], [79], [106]. Moreover, MPC research considers different assumptions regarding adversarial behavior such as the well-known semi-honest [47], [28] and malicious security model [42], [71], [73], [72], [29]) as well as numbers of corrupted computation parties (e.g., honest majority [59], [61] or dishonest majority/full threshold security [41], [73], [28], [105], [47]). 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 [1], [130], [6].

D. 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 [8], and in privacy-preserving machine learning such as for division [33], activation functions [98], [63], [30], and completely changing the classifier to be MPC-friendly [111]. Also, for FHE, approximations are used such as in the approximate HE scheme CKKS [37], which is implemented in the HEAAN library[10] and was used for approximate genomic computations in [74]. 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.

E. Secure Aggregation

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., [50], [81]. It has come a long way since then, resulting in practical solutions for real-world applications nowadays.

For example, Prio [39] 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 [39] with a Boolean secret sharing-based client input validation and an additional conversion from Boolean to arithmetic sharing. Similar to our work, it has a multi-server setup to jointly compute statistical functions on private inputs. Compared to Prio+ [2], we optimize the naive bit-to-arithmetic conversion presented in [2] for our $\mathcal{F}_{\mathsf{SecAgg}}$ protocol (cf. §III-A), resulting in reduced communication cost of $2.4\times$ with exact results and $4\times$ with our novel approximating variant for $\mathsf{n} = 10^5$, where n is the number of clients. Popa et al. [108] specifically focus on secure location-based aggregation statistics, Joye et al. [66] on time-series data, and PrivEx [49] on traffic data in anonymous communication.

So et al. [121] point out that differences among securely aggregated updates across multiple training iterations can also leak information about the contribution of individual clients. Most existing secure aggregation schemes are executed on one training iteration, i.e., they cannot protect against multi-round attacks. An exception is POSEIDON [116] which runs FL fully under encryption, but at the cost of significant computational overhead on clients' and server's side. Instead, So et al. [121] propose to organize clients in batches that can only be chosen together for a training iteration. This approach is orthogonal and fully compatible with ScionFL.

F. Poisoning Attacks & Defenses

Poisoning attacks can be categorized into untargeted and targeted attacks based on the goals of the attacker [52]. In the former case, the attacker aims to corrupt the global model so that it reduces or even destroys the performance of the trained

[10] https://github.com/snucrypto/HEAAN

model for a large number of test inputs, yielding a final global model with a high error rate [52], [118], [11]. 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 [20], [124]. 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* [10]. As discussed in §IV, we consider only untargeted poisoning following the argument in [119]: 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 [11], Fang [52], and Shejwalkar et al. [118], which are most relevant to our work.

– *Little is Enough (LIE) attack* [11]: In LIE [11], malicious clients manipulate their local updates by adding noise drawn from the normal distribution to "clean" updates they created following the normal training process to cause a disorientation. LIE assumes independent and identically distributed (iid) data and was tested against various robust aggregations such as trimmed-mean [139].

– *Fang et al.* [52]: The authors of [52] 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 [118].

– *Shejwalkar and Houmansadr* [118]: The attacks of [118] follow a similar idea as [52]: 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 following "Min-Max" optimization problem:

$$\underset{\gamma}{\operatorname{argmax}} \max_{i \in [n]} \|\nabla^m - \nabla_i\|_2 \leq \max_{i,j \in [n]} \|\nabla_i - \nabla_j\|_2 \qquad (13)$$

$$\nabla^m = \mathsf{f}_{\mathsf{avg}}(\nabla_{\{i \in [n]\}}) + \gamma \nabla^p, \qquad (14)$$

where $\mathsf{f}_{\mathsf{avg}}(\nabla_{\{i \in [n]\}})$ is the average gradient and $\gamma\ \nabla^p$ is the adversary's perturbation vector, i.e., either the inverse unit vector of the (simulated) benign gradients, the inverse average standard deviations, or the average gradient with flipped sign of all updates. For details, we refer to §IV in [118].

Note that although the authors of [118] suggest several flavours of their attack based on different levels of adversarial knowledge, we compare to their Min-Max attack as it (i) does not make the unrealistic assumption that an adversary knows defenses in place and (ii) it is more destructive than LIE [11] for almost all datasets [118]. We do not consider Fang et al. [52]'s attack as it requires the guess of the robust aggregation rule, i.e., defense mechanism, which is unrealistic in a real-world deployment. Taking those considerations into account, we evaluate the robustness of ScionFL-Aura against the state-of-the-art Min-Max attack of [118] in §IV-B.

Poisoning Defenses: Simple parameter-wise averaging is very sensitive to outliers and, thus, can easily hamper accuracy. Therefore, Byzantine-robust defenses aim to make FL robust against (untargeted) attacks. To do so, Krum [21] selects only one local update, namely the one with the closest $n - m - 2$ local updates as update for the global model, where n is the number of clients and m the number of anticipated malicious clients. Multi-krum [21] extends this idea to a selection of c (instead of just one) updates. Median [139] is an 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 [32], [48], [85]. FLTrust [32] and FLOD [48] use the ReLU-clipped cosine-similarity/Hamming distance between each received update and the aggregator-computed baseline update based on the auxiliary dataset. FLDetector [141] detects malicious clients by checking their model updates' consistency based on historical model updates. RSA [85] uses an L_1-norm-based regularization, which is also comparing to the aggregator-computed baseline update. The recently proposed Divider and Conquer (DnC) aggregation [118] 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 straight-forward manner or lead to an intolerable overhead. Only two works, namely FLAME [102] and BaFFLe [5] simultaneously consider both threats. Concretely, FLAME [102] uses a density-based clustering to remove updates with significantly different cosine distances (i.e., different directions) combined with clipping (for more subtle manipulations). 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.

Recently, ELSA [110] considered a distributed aggregator setup and proposed methods to address poisoning attacks from malicious clients. However, ELSA's defense methods are designed to work independently on the gradients of each client, specifically using L_2 and L_∞ norms. ELSA does not support defenses such as trimmed mean, median, or Krum [21], as already mentioned in their work. Consequently, ELSA's defense mechanism is not sufficiently robust to guard against stronger attacks like Min-Max. The defenses against these types of attacks require collective information about the gradients instead of treating each gradient individually. To illustrate this point, we conducted an evaluation of ELSA against the Min-Max attack with 10% corruption on a three-layer Convolutional Neural Network using the FashionMNIST dataset. Even after 1000 epochs of training, we observed a significant drop in accuracy to below 70%.

G. Global Model Privacy

Secure aggregation addresses the concern of the aggregator observing individual model updates in the clear, potentially leading to the leakage of private information (cf. §I-B). However, existing works (e.g., [91]) have noted that even from the aggregated global model (computed via secure aggregation but distributed in the clear), attackers can deduce private information of individual clients, e.g., through model inversion attacks [133]. To mitigate such issues, one can apply orthogonal techniques such as differential privacy on top of ScionFL [69], [103]. Additionally, there are works like HyFL [92] proposing a framework to ensure full model privacy in FL, but, they do not consider communication-efficient secure aggregation, as in ScionFL.

Appendix B
Preliminaries

This section provides relevant details regarding the primitives used in this work. We begin with providing the necessary MPC background and protocols. 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. Since we consider dishonest majority setting to work with, we utilize the (semi-honest variant of) primitives from [45], [41], [42], [17] in a black-box manner.

A. MPC Protocols

In this section, we go over the details of the underlying MPC protocols used in our scheme. We consider three MPC servers, $\mathcal{S} = \{\mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3\}$, to which the clients delegate the aggregation computation, as shown in Fig. 1. All the operations are carried out in either an ℓ-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 [87], [17], [125] discussed in §III, starting with the sharing semantics.

Sharing Semantics: We use two different sharing schemes:

1) $[\cdot]$-*sharing*. A value $\mathsf{v} \in \mathbb{Z}_{2^\ell}$ is said to be $[\cdot]$-shared among MPC servers in $\mathcal{S}$, if each server S_i, for $i \in [3]$, holds $\mathsf{v}_i \in \mathbb{Z}_{2^\ell}$ such that $\mathsf{v}_1 + \mathsf{v}_2 + \mathsf{v}_3 = \mathsf{v}$.

2) $\langle\cdot\rangle$-*sharing*. In this sharing, every $\mathsf{v} \in \mathbb{Z}_{2^\ell}$ is associated with two values: a random mask $\lambda_{\mathsf{v}} \in \mathbb{Z}_{2^\ell}$ and a masked value $\mathsf{m}_{\mathsf{v}} \in \mathbb{Z}_{2^\ell}$, such that $\mathsf{v} = \mathsf{m}_{\mathsf{v}} + \lambda_{\mathsf{v}}$. Here, the share of an MPC server is defined as a tuple of the form $(\mathsf{m}_{\mathsf{v}}, [\lambda_{\mathsf{v}}])$.

Handling Decimal Values: The MPC protocol we use is designed over a ring architecture, while the underlying FL algorithms handle decimal numbers. To address this compatibility issue, we employ the well-known Fixed-Point Arithmetic (FPA) technique [33], [98], [97], which encodes a decimal number in ℓ-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 $\mathsf{f} = 16$ in this work.

We will now go over the MPC protocols used in our scheme. We assume that the protocols' inputs are in $\langle\cdot\rangle$-shared form, and that the output is generated in $\langle\cdot\rangle$-shared form among the MPC servers.

Inner Product Computation: For simplicity, consider the multiplication of two values $x, y \in \mathbb{Z}_{2^\ell}$ as per the $\langle\cdot\rangle$-sharing semantics. We have

$$\begin{aligned} z = xy &= (\mathsf{m}_x + \lambda_y)(\mathsf{m}_x + \lambda_y) \\ &= \mathsf{m}_x\mathsf{m}_y + \mathsf{m}_x\lambda_y + \mathsf{m}_y\lambda_x + \lambda_x\lambda_y. \end{aligned}$$

Since the λ values are independent of the underlying secret, the servers can compute $[\cdot]$-shares of the term $\lambda_x\lambda_y$ during preprocessing using the $\Pi_{\mathsf{IP}}^{\mathsf{Pre}}()$ protocol [41], [17]. 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 [33], [98] must be performed in order to keep the result z in FPA format after a multiplication. For this, we use the truncation pair method [97], wherein a tuple of the form $(r, r/2^{\mathsf{f}})$ is generated in $\langle\cdot\rangle$-shared form among the servers during preprocessing using the $\Pi_{\mathsf{Tr}}()$ protocol [42]. Then, with very high probability, we have

$$z/2^{\mathsf{f}} = (z - r)/2^{\mathsf{f}} + r/2^{\mathsf{f}}.$$

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

For the case of the inner-product computation (Fig. 13), the task can be divided 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 [105].

Protocol $\Pi_{\mathsf{IP}}(\langle\vec{\mathbf{X}}_{\mathsf{d}\times 1}\rangle, \langle\vec{\mathbf{Y}}_{\mathsf{d}\times 1}\rangle, \mathsf{f})$

Preprocessing:

1. Execute $\Pi_{\mathsf{IP}}^{\mathsf{Pre}}([\lambda_{\vec{\mathbf{X}}}], [\lambda_{\vec{\mathbf{Y}}}])$ to obtain $[\gamma_z]$ with $\gamma_z = \lambda_{\vec{\mathbf{X}}} \odot \lambda_{\vec{\mathbf{Y}}}$.
2. Execute $\Pi_{\mathsf{Tr}}()$ to generate $([r], \langle r/2^{\mathsf{f}}\rangle)$.

Online:

1. S_j, for $j \in [\tau]$, locally computes as follows ($\Delta = 1$ if $j = 1$, else 0):
- $[(z - r)]_j = \Delta \cdot (\mathsf{m}_{\vec{\mathbf{X}}} \odot \mathsf{m}_{\vec{\mathbf{Y}}}) + \mathsf{m}_{\vec{\mathbf{X}}} \odot [\lambda_{\vec{\mathbf{Y}}}]_j + \mathsf{m}_{\vec{\mathbf{Y}}} \odot [\lambda_{\vec{\mathbf{X}}}]_j + [\gamma_z]_j - [r]_j$.

2. S_j, for $j \in [\tau]$, sends $[(z - r)]_j$ to S_1, who computes $(z - r)$ and sends to all the servers.
3. Locally compute $\langle z\rangle = \langle (z - r)/2^{\mathsf{f}}\rangle + \langle r/2^{\mathsf{f}}\rangle$.

Fig. 13: Inner product protocol.

Bit-to-Arithmetic Protocol: Given the Boolean sharing of $\mathsf{b} \in \mathbb{Z}_2$, protocol Π_{BitA} computes the arithmetic sharing of the bit b over $\mathbb{Z}_{2^\ell}$. As shown in Eq. 15, the arithmetic equivalent $\tilde{\mathsf{b}}$ for a bit $\mathsf{b} = \mathsf{m}_{\mathsf{b}} \oplus \lambda_{\mathsf{b}}$ can be obtained as

$$\tilde{\mathsf{b}} = \mathsf{m}_{\mathsf{b}} \oplus \lambda_{\mathsf{b}} = \mathsf{M}_{\mathsf{b}} + (1 - 2\mathsf{m}_{\mathsf{b}}) \cdot \Lambda_{\mathsf{b}}. \qquad (15)$$

Here, M_{b} and Λ_{b} denote the arithmetic equivalents of m_{b} and λ_{b} respectively. In our protocol shown in Fig. 14, MPC servers invoke $\Pi_{\mathsf{BitA}}^{\mathsf{Pre}}$ protocol [42], [105] on the Boolean $[\cdot]$-shares of λ_{b} in the preprocessing phase to obtain its respective arithmetic shares. This enables the servers to locally compute an additive sharing of $\tilde{\mathsf{b}}$ during the online phase, as shown above. The rest of the steps proceed similar to the inner-product protocol and we omit the details.

Protocol $\Pi_{\mathsf{BitA}}(\langle \mathsf{b} \rangle^{\mathbf{B}})$

Preprocessing:

1. Execute $\Pi^{\mathsf{Pre}}_{\mathsf{BitA}}([\lambda_{\mathsf{b}}]^{\mathbf{B}})$ to obtain $[\lambda_{\mathsf{b}}]$.
2. Locally generate $([r], \langle r \rangle)$ for a random $r \in \mathbb{Z}_{2^\ell}$.

Online:

1. S_j, for $j \in [\tau]$, locally computes as follows ($\Delta = 1$ if $j = 1$, else 0):
- $[(z - r)]_j = \Delta \cdot \mathsf{m_b} + (1 - 2\mathsf{m_b}) \cdot [\lambda_{\mathsf{b}}]_j - [r]_j$.

2. S_j, for $j \in [\tau]$, sends $[(z - r)]_j$ to S_1, who computes $(z - r)$ and sends to all the servers.
3. Locally compute $\langle z \rangle = \langle (z - r) \rangle + \langle r \rangle$.

Fig. 14: Bit-to-arithmetic conversion protocol.

To instantiate $\Pi^{\mathsf{Pre}}_{\mathsf{BitA}}$, we use SPDZ-style computations [71], [113], where oblivious transfer (OT) instances [9], [27], [40] are used among every pair of servers. Let $\Pi^{\mathsf{ij}}_{\mathsf{OT}}$ denote an instance of 1-out-of-2 OT with $\mathsf{S_i}$ being the sender and $\mathsf{S_j}$ being the receiver. Here, $\mathsf{S_i}$ inputs the sender messages (x_0, x_1) while $\mathsf{S_j}$ inputs the receiver choice bit $c \in \mathbb{Z}_2$ and obtains x_c as the output, for $x_0, x_1 \in \mathbb{Z}_{2^\ell}$.

Protocol $\Pi^{\mathsf{Pre}}_{\mathsf{BitA}}([\mathsf{b}]^{\mathbf{B}})$

OT Instance - I: $[\mathsf{b}]_1[\mathsf{b}]_2$

1. S_1 samples random $r_{12} \in \mathbb{Z}_{2^\ell}$.
2. S_1 and S_2 executes $\Pi^{12}_{\mathsf{OT}}((r_{12}, r_{12} + [\mathsf{b}]_1), [\mathsf{b}]^{\mathbf{B}}_2)$.
3. S_1 sets $y^1_{12} = -r_{12}$ and S_2 sets the OT output as y^2_{12}.

OT Instances - II & III: $[\mathsf{b}]_1[\mathsf{b}]_3$, $[\mathsf{b}]_2[\mathsf{b}]_3$

These are similar to the computation of $[\mathsf{b}]_1[\mathsf{b}]_2$ discussed above.

OT Instances - IV & IV: $[\mathsf{b}]_1[\mathsf{b}]_2[\mathsf{b}]_3$

1. Computation can be broken down to $([\mathsf{b}]_1[\mathsf{b}]_2) \cdot [\mathsf{b}]_3 = (y^1_{12} + y^2_{12}) \cdot [\mathsf{b}]_3$.
2. Execute Π^{13}_{OT} for $y^1_{12} \cdot [\mathsf{b}]^{\mathbf{B}}_3$. Let z^1_{13} and z^2_{13} denote the respective shares of S_1 and S_3.
3. Execute Π^{23}_{OT} for $y^2_{12} \cdot [\mathsf{b}]^{\mathbf{B}}_3$. Let z^1_{23} and z^2_{23} denote the respective shares of S_2 and S_3.

Computation of final shares

S_1: $[\mathsf{b}]_1 = \mathsf{b}_1 - 2y^1_{12} - 2y^1_{13} + 4z^1_{13}$.
S_2: $[\mathsf{b}]_2 = \mathsf{b}_2 - 2y^2_{12} - 2y^1_{23} + 4z^1_{23}$.
S_3: $[\mathsf{b}]_3 = \mathsf{b}_3 - 2y^2_{13} - 2y^2_{23} + 4z^2_{13} + 4z^2_{23}$.

Fig. 15: Bit-to-arithmetic preprocessing.

To generate the arithmetic sharing of λ_{b} from its Boolean shares in $[\cdot]$-shared form, a simple method would be to apply a 3-XOR using a daBit-style approach [113], 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 [113]. Since Prio+ operates over two MPC servers, we extend their optimized daBit-generation protocol (cf. [2, $\mathsf{daBitGen}_p$]) to our setting with three servers.

Given two bits $\mathsf{b_i}, \mathsf{b_j} \in \mathbb{Z}_2$, the arithmetic share corresponding to their product can be generated using one instance of $\Pi^{\mathsf{ij}}_{\mathsf{OT}}$ with $(x_0 = r, x_1 = r + \mathsf{b_i})$ as the OT-sender messages and $\mathsf{b_j}$ as the OT-receiver choice bit. With this observation and using Eq. 8, servers can compute $[\cdot]$-shares corresponding to the bit λ_{b} using five OT invocations. The formal details appear in Fig. 15.

Protocol $\Pi^{\mathsf{sum}}_{\mathsf{BitA}}(\langle \vec{\mathbf{M}}_{\mathsf{d}\times 1} \rangle^{\mathbf{B}})$

Preprocessing:

1. Execute $\Pi^{\mathsf{Pre}}_{\mathsf{BitA}}([\lambda_{\vec{\mathbf{M}}}]^{\mathbf{B}})$ to obtain $[\lambda_{\vec{\mathbf{M}}}]$.
2. Locally generate $([r], \langle r \rangle)$ for a random $r \in \mathbb{Z}_{2^\ell}$.

Online:

1. S_j, for $j \in [\tau]$, locally computes as follows ($\Delta = 1$ if $j = 1$, else 0):
- $[(z - r)]_j = \Delta \cdot \mathsf{Agg\text{-}R}(\mathsf{m}_{\vec{\mathbf{M}}}) + (1 - 2\mathsf{m}_{\vec{\mathbf{X}}}) \odot [\lambda_{\vec{\mathbf{Y}}}]_j - [r]_j$.

2. S_j, for $j \in [\tau]$, sends $[(z - r)]_j$ to S_1, who computes $(z - r)$ and sends to all the servers.
3. Locally compute $\langle z \rangle = \langle (z - r) \rangle + \langle r \rangle$.

Fig. 16: Bit-to-arithmetic sum protocol.

For the case of approximate bit conversion discussed in §III-B, the number of OT instances can be further reduced to three following Eq. 10. Concretely, the conversion involves computation of just $[\mathsf{b}]_1[\mathsf{b}]_2[\mathsf{b}]_3$ and hence the OT instances II & III described in Fig. 15 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 $\Pi^{\mathsf{sum}}_{\mathsf{BitA}}$ is given in Fig. 16.

Bit Injection Protocol: Given a bit $\mathsf{b} = \mathsf{m_b} \oplus \lambda_{\mathsf{b}}$ and $\mathsf{s} = \mathsf{M_s} + \Lambda_{\mathsf{s}}$, the bit injection operation involves computing the value $\mathsf{b} \cdot \mathsf{s}$ that can be obtained as

$$\begin{aligned} \mathsf{b} \cdot \mathsf{s} &= (\mathsf{M_b} + (1 - 2\mathsf{m_b}) \cdot \Lambda_{\mathsf{b}}) \cdot (\mathsf{M_s} + \Lambda_{\mathsf{s}}) \\ &= \mathsf{M_b M_s} + \mathsf{M_b}\Lambda_{\mathsf{s}} + (1 - 2\mathsf{m_b}) \cdot (\Lambda_{\mathsf{b}}\mathsf{M_s} + \Lambda_{\mathsf{b}}\Lambda_{\mathsf{s}}). \quad (16) \end{aligned}$$

Given a boolean vector $\vec{\mathbf{M}}_{\mathsf{d}\times 1}$ and an arithmetic vector $\vec{\mathbf{N}}_{\mathsf{d}\times 1}$ in the secret-shared form, protocol Π_{BI} computes the inner product of the two vectors, defined as $z = \vec{\mathbf{M}} \odot \vec{\mathbf{N}}$. This protocol is similar to the inner product protocol Π_{IP} (Fig. 13), with the main difference being that $\vec{\mathbf{M}}$ is a boolean vector.

During the preprocessing, servers first generate the arithmetic shares of $\lambda_{\vec{\mathbf{M}}}$ from its boolean shares, similar to the bit-to-arithmetic protocol Π_{BitA} in Fig. 14. In this case, $\Pi^{\mathsf{Pre}}_{\mathsf{BI}}$ is same as the $\Pi^{\mathsf{Pre}}_{\mathsf{IP}}$ primitive discussed in Fig. 13. The remaining steps are similar to the Π_{IP} in Fig. 13 and we omit the details.

Protocol $\Pi_{\mathsf{BI}}(\langle \vec{\mathbf{M}}_{\mathsf{d}\times 1} \rangle^{\mathbf{B}}, \langle \vec{\mathbf{N}}_{\mathsf{d}\times 1} \rangle, \mathsf{f})$

Preprocessing:

1. Execute $\Pi^{\mathsf{Pre}}_{\mathsf{BitA}}([\lambda_{\vec{\mathbf{M}}}]^{\mathbf{B}})$ to obtain $[\lambda_{\vec{\mathbf{M}}}]$.
2. Execute $\Pi^{\mathsf{Pre}}_{\mathsf{BI}}([\lambda_{\vec{\mathbf{M}}}], [\lambda_{\vec{\mathbf{N}}}])^a$ to obtain $[\gamma_{\vec{\mathbf{Q}}}]$ with $\gamma_{\vec{\mathbf{Q}}} = \lambda_{\vec{\mathbf{M}}} \circ \lambda_{\vec{\mathbf{N}}}$.
3. Execute $\Pi_{\mathsf{Tr}}()$ to generate $([r], \langle r/2^{\mathsf{f}} \rangle)$.

Online:

1. S_j, for $j \in [\tau]$, locally computes as follows ($\Delta = 1$ if $j = 1$, else 0):
- $T^1_j = \Delta \cdot (\mathsf{m}_{\vec{\mathbf{M}}} \odot \mathsf{m}_{\vec{\mathbf{N}}}) \ + \ \mathsf{m}_{\vec{\mathbf{M}}} \odot [\lambda_{\vec{\mathbf{N}}}]_j$.
- $T^2_j = ((1 - 2\mathsf{m}_{\vec{\mathbf{M}}}) \circ \mathsf{m}_{\vec{\mathbf{N}}}) \odot [\lambda_{\vec{\mathbf{M}}}]_j \ + \ (1 - 2\mathsf{m}_{\vec{\mathbf{M}}}) \odot [\gamma_{\vec{\mathbf{Q}}}]_j$.
- $[(z - r)]_j = T^1_j + T^2_j - [r]_j$.

2. S_j, for $j \in [\tau]$, sends $[(z - r)]_j$ to S_1, who computes $(z - r)$ and sends to all the servers.
3. Locally compute $\langle z \rangle = \langle (z - r)/2^{\mathsf{f}} \rangle + \langle r/2^{\mathsf{f}} \rangle$.

[a] $\Pi^{\mathsf{Pre}}_{\mathsf{BI}}$ is same as $\Pi^{\mathsf{Pre}}_{\mathsf{IP}}$ (Fig. 13) in the setting considered in this work.

Fig. 17: Bit injection (sum) protocol.

B. Binomial Sum

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

1) $\sum\limits_{p=0}^{n} p \cdot \binom{n}{p} = n \cdot 2^{n-1}$.

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

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

$$\sum_{p=0}^{n} \binom{n}{p} y^p = (1+y)^n \tag{17}$$

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

$$\sum_{p=0}^{n} \binom{n}{p} p \cdot y^{p-1} = n \cdot (1+y)^{n-1} \tag{18}$$

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

$$\sum_{p=0}^{n} (-1)^{p-1} p \cdot \binom{n}{p} = 0 \tag{19}$$

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

C. 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 ≈ 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., [129]). To summarize, we state these resulting overheads in Tab. VI.

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

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

APPENDIX C
SCIONFL: ADDITIONAL DETAILS

This section provides addition details of our FL framework ScionFL presented in §III. We begin with providing additional details regarding the approximate bit conversion discussed in §III-B.

A. Multi-bit Quantization Schemes

This section describes how our scheme ScionFL can be extended to support multi-bit linear quantization schemes, in which each coordinate is classified into more than two levels, resulting in each coordinate being represented by more than a single bit.

For instance, consider the quantization in TernGrad [134], where each coordinate is compressed to one of the three levels $\{-1, 0, 1\}$. Here, each coordinate can be represented using two bits, say b_1 and b_2 and the quantized level can be computed as $2\mathsf{b}_1 - \mathsf{b}_2$.

To use our scheme, each client C_i share the bits separately using the underlying boolean secret sharing scheme, i.e., $\langle \mathsf{b}_1 \rangle_i^{\mathbf{B}}$ and $\langle \mathsf{b}_2 \rangle_i^{\mathbf{B}}$. MPC servers use our instantiations of $\mathcal{F}_{\mathsf{SecAgg}}$ functionality discussed in §III-A to aggregate each of the bits and obtain the result in arithmetic sharing format, i.e, $\langle \mathsf{b}_1 \rangle$ and $\langle \mathsf{b}_2 \rangle$. The final result can be locally computed by the MPC servers as $2\langle \mathsf{b}_1 \rangle + \langle \mathsf{b}_2 \rangle$, since the underlying MPC protocol used in ScionFL is linear.

B. Approximate Bit Conversion

Lemma III.1 (Expected Values). *Given a bit* $\mathsf{b} = \oplus_{i=1}^{\mathsf{q}} \mathsf{b}_i$ *and* $b = \mathsf{term_s} + \mathsf{term_m} + \mathsf{term_p}$ *with*

$$\mathsf{term_s} = \sum_{\{\mathsf{b}_e\} \in \mathcal{Q}^{|1|}} \tilde{\mathsf{b}}_e, \quad \mathsf{term_m} = \sum_{k=2}^{\mathsf{q}-1} (-2)^{k-1} \sum_{\{\mathsf{b}_{e_1}, \ldots, \mathsf{b}_{e_k}\} \in \mathcal{Q}^{|k|}} \tilde{\mathsf{b}}_{e_1} \tilde{\mathsf{b}}_{e_2} \ldots \tilde{\mathsf{b}}_{e_k},$$

$$\mathsf{term_p} = (-2)^{\mathsf{q}-1} \prod_{\mathsf{b}_e \in \mathcal{Q}} \tilde{\mathsf{b}}_e,$$

we have $\mathbb{E}[\mathsf{term_s} \mid \mathsf{b}] = \mathsf{q}/2$, $\mathbb{E}[\mathsf{term_m} \mid \mathsf{b}] = (\mathsf{q}\text{-}1) \bmod 2 - \mathsf{q}/2$, *and* $\mathbb{E}[\mathsf{term_p} \mid \mathsf{b}] = \mathsf{b} - (\mathsf{q}\text{-}1) \bmod 2$.

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 $\mathsf{b} = 0$, while the other half correspond to $\mathsf{b} = 1$. The truth table for three shares ($\mathsf{q} = 3$) is given in Tab. VII as a reference.

b	b_1	b_2	b_3	$\mathsf{term_s}$	$\mathsf{term_m}$	$\mathsf{term_p}$	$\tilde{\mathsf{b}}$
0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	1
1	0	1	0	1	0	0	1
0	0	1	1	2	-2	0	0
1	1	0	0	1	0	0	1
0	1	0	1	2	-2	0	0
0	1	1	0	2	-2	0	0
1	1	1	1	3	-6	4	1

TABLE VII: Truth table for $\mathsf{b} = \mathsf{b}_1 \oplus \mathsf{b}_2 \oplus \mathsf{b}_3$. The rows corresponding to $\mathsf{b} = 0$ are highlighted. $\tilde{\mathsf{b}}$ denotes the arithmetic equivalent of b.

[11] The size of the last chunk is kept above some threshold, e.g., 2^9 to keep the overhead of the scales small.

Sum Term ($\mathsf{term_s}$)*:* For each row of the form $(\mathsf{b}_1,\ldots,\mathsf{b}_\mathsf{q})$ in T_b, $\mathsf{term_s}$ equals $\tilde{\mathsf{b}}_1+\ldots+\tilde{\mathsf{b}}_\mathsf{q}$, which can be interpreted as the number of $\tilde{\mathsf{b}}_i$'s selected out of the q possible. Furthermore, there are a total of $\binom{\mathsf{q}}{k}$ rows with sums equal to k, with k being odd corresponding to the row for $\mathsf{b}=1$ and k being even corresponding to the row for $\mathsf{b}=0$. As a result, given $\mathsf{b}=0$, the expectation of the sum term can be calculated as the product of $1/2^{\mathsf{q}-1}$ (corresponding to rows in T_b with $\mathsf{b}=0$) and the sum of terms of the form $k\cdot\binom{\mathsf{q}}{k}$ with k being even. Using Lem. B.1 in §B, we get

$$\mathbb{E}[\mathsf{term_s} \mid (\mathsf{b}=0)] = \frac{1}{2^{\mathsf{q}-1}}\cdot\sum_{k=0}^{\lfloor \mathsf{q}/2\rfloor} 2k\binom{\mathsf{q}}{2k} = \frac{1}{2^{\mathsf{q}-1}}\cdot\mathsf{q}\cdot 2^{\mathsf{q}-2} = \mathsf{q}/2.$$

Similarly, we obtain $\mathbb{E}[\mathsf{term_s} \mid (\mathsf{b}=1)] = \mathsf{q}/2$. To summarize, we have $\mathbb{E}[\mathsf{term_s} \mid \mathsf{b}] = \mathsf{q}/2$.

Product Term ($\mathsf{term_p}$)*:* The product of all the q shares will be 1 only if all the shares are 1, otherwise it will be 0. Moreover, all shares of b being 1 correspond to $\mathsf{b}=1$ if q is odd, and $\mathsf{b}=0$ otherwise. Now, when q is odd then $\mathsf{term_p}=(-2)^{\mathsf{q}-1}$ with probability $\frac{1}{2^{\mathsf{q}-1}}$ (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

$$\begin{aligned}
\mathbb{E}[\mathsf{term_p} \mid (\mathsf{b}=0 \land \mathsf{q}\text{ is odd})] &= \frac{1}{2^{\mathsf{q}-1}}\cdot 0 &&= 0\\
\mathbb{E}[\mathsf{term_p} \mid (\mathsf{b}=1 \land \mathsf{q}\text{ is odd})] &= \frac{1}{2^{\mathsf{q}-1}}\cdot(-2)^{\mathsf{q}-1} &&= 1
\end{aligned}$$

Similarly, the case for even q can be written as

$$\begin{aligned}
\mathbb{E}[\mathsf{term_p} \mid (\mathsf{b}=0 \land \mathsf{q}\text{ is even})] &= \frac{1}{2^{\mathsf{q}-1}}\cdot(-2)^{\mathsf{q}-1} &&= -1\\
\mathbb{E}[\mathsf{term_p} \mid (\mathsf{b}=1 \land \mathsf{q}\text{ is even})] &= \frac{1}{2^{\mathsf{q}-1}}\cdot 0 &&= 0
\end{aligned}$$

The above observation can be summarized as $\mathbb{E}[\mathsf{term_p} \mid \mathsf{b}] = \mathsf{b} - (\mathsf{q}\text{-}1) \bmod 2$.

Middle Term ($\mathsf{term_m}$)*:* Given $\mathbb{E}[\mathsf{b}]=\mathsf{b}$, $\mathbb{E}[\mathsf{term_s} \mid \mathsf{b}]$ and $\mathbb{E}[\mathsf{term_p} \mid \mathsf{b}]$, the expectation of $\mathsf{term_m}$ can be calculated as

$$\begin{aligned}
\mathbb{E}[\mathsf{term_m} \mid \mathsf{b}] &= \mathbb{E}[\mathsf{b}] - \mathbb{E}[\mathsf{term_s} \mid \mathsf{b}] - \mathbb{E}[\mathsf{term_p} \mid \mathsf{b}]\\
&= \mathsf{b} - \mathsf{q}/2 - (\mathsf{b} - (\mathsf{q}\text{-}1) \bmod 2) = (\mathsf{q}\text{-}1) \bmod 2 - \mathsf{q}/2.
\end{aligned}$$

This concludes the proof of Lem. III.1. □

Efficiency Analysis: We measure the efficiency gains achieved by our approximation method, discussed in §III-B, by counting the number of *cross terms*[12] 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.[13]

Computation	#cross-terms	
	Exact ($\tilde{\mathsf{b}}$)	Approximate ($\hat{\mathsf{b}}$)
Bit-to-Arithmetic	$2^\mathsf{q}-\mathsf{q}-1$	1
Bit Injection	$2^\mathsf{q}+\mathsf{q}^2-2\mathsf{q}-1$	$\mathsf{q}^2-\mathsf{q}+1$

TABLE VIII: Efficiency analysis via approximate bit conversion with respect to the #cross-terms involved.

Tab. VIII provides details regarding the number of cross terms involved in obtaining the arithmetic equivalent of $\mathsf{b}=\oplus_{i=1}^{\mathsf{q}}\mathsf{b}_i$. The gains increase significantly with a higher number of shares q due to the exponential growth in the number of cross terms for the exact computation. Tab. VIII also provides details for a bit injection operation in which the product of a Boolean bit b and a scale value s is securely computed. Given $\mathsf{s}=\sum_{i=1}^{\mathsf{q}}\mathsf{s}_i$, the value $\mathsf{b}\cdot\mathsf{s}$ can be computed by first computing either $\tilde{\mathsf{b}}$ or $\hat{\mathsf{b}}$ (depending on whether an exact or approximate value is required) and then multiplying by s.

C. ScionFL-Aura: Additional Details

In this subsection, we provide additional details of our ScionFL-Aura.

Sub-protocols: Here, we provide the details of the sub-protocols used in ScionFL-Aura (cf. Alg. 1 in §IV-A).

Algorithm 2 Quantized Aggregation

```
procedure AGGREGATE($\{\vec{\sigma}_{Y_i}, \mathsf{s}_{Y_i}^{min}, \mathsf{s}_{Y_i}^{max}\}_{i\in\alpha}$)
    $\vec{Z} \leftarrow \vec{0}$
    for $k \leftarrow 1$ to $\alpha$ do
        $\vec{Z} \leftarrow \vec{Z} + \left(\mathsf{s}_{Y_k}^{min} \oplus \vec{\sigma}_{Y_k} \circ (\mathsf{s}_{Y_k}^{max} - \mathsf{s}_{Y_k}^{min})\right)$
    end for
    $\vec{Z} \leftarrow \vec{Z}/\alpha$
    return $\vec{Z}$
end procedure
```

Alg. 2 computes the aggregation of α quantized vectors. As shown in Eq. 4, the dequantized value of a vector $\vec{Y}$, given its quantized form $(\vec{\sigma}_Y, \mathsf{s}_Y^{min}, \mathsf{s}_Y^{max})$, can be computed as

$$\vec{Y} = \mathsf{s}_Y^{min} \;\oplus\; \vec{\sigma}_Y \circ (\mathsf{s}_Y^{max} - \mathsf{s}_Y^{min}).$$

The above operation essentially places s_Y^{min} in those positions of the vector $\vec{Y}$ with the corresponding bit in $\vec{\sigma}_Y$ being zero, and the rest with s_Y^{max}.

Algorithm 3 L2-Norm Computation (Quantized)

```
procedure L2-NORMQ($\vec{\sigma}_Y, \mathsf{s}_Y^{min}, \mathsf{s}_Y^{max}$)
    $\beta \leftarrow$ LEN($\vec{\sigma}_Y$) // Dimension of $\vec{\sigma}_Y$
    $N_O \leftarrow$ SUM($\vec{\sigma}_Y$) // Number of ones in $\vec{\sigma}_Y$
    $N_Z \leftarrow \beta - N_O$ // Number of zeros in $\vec{\sigma}_Y$
    return $\sqrt{N_Z \cdot (\mathsf{s}_Y^{min})^2 + N_O \cdot (\mathsf{s}_Y^{max})^2}$
end procedure
```

Alg. 3 computes the L_2-norm of a quantized vector. As discussed in §II-D, a quantized vector $\vec{Y}_\sigma$ consists of a binary vector $\vec{\sigma}_Y$ and the respective min. and max. scales $\mathsf{s}_Y^{min}/\mathsf{s}_Y^{max}$. In this case, we observe that the squared L_2-norm can be obtained by first counting the number of zeroes and ones in

[12] Terms for which interaction among MPC servers is necessary.

[13] We acknowledge that the analysis cannot provide an exact comparison, owing to the presence of the product term in the approximation. e.g., depending on the underlying MPC setup, the product term ($\mathsf{term_p}$) may require more communication than the middle terms ($\mathsf{term_m}$), and therefore the effect of approximation may be minimized.

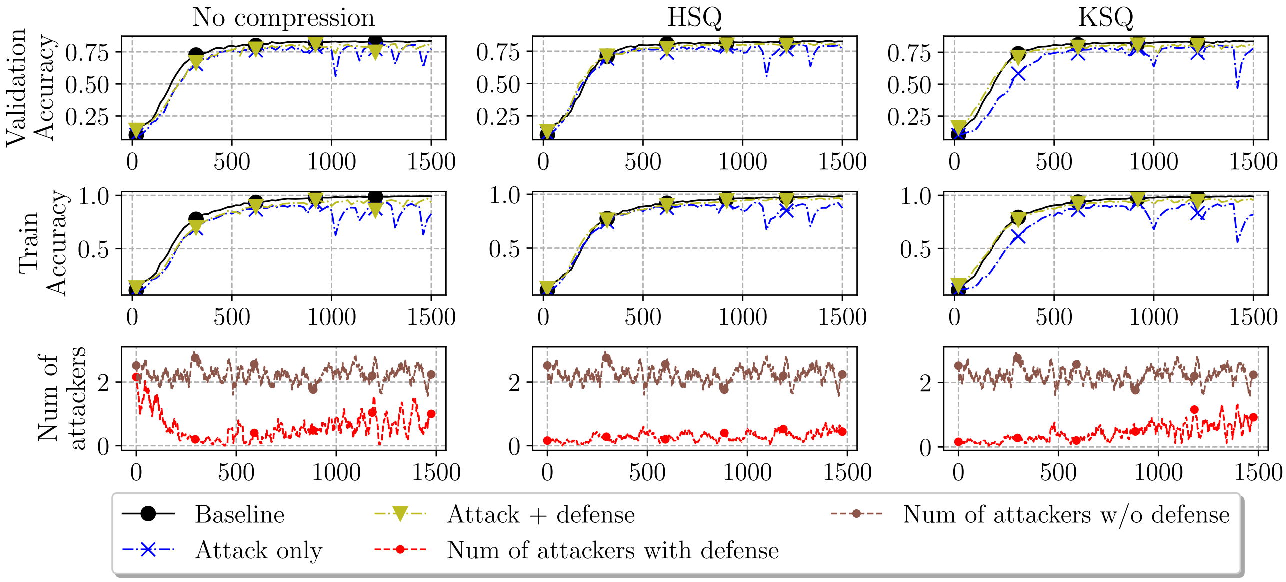


Fig. 18: Effect of *Min-Max* attack [118] on training VGG11 with CIFAR10 for 1500 aggregation rounds with and without our defense ScionFL-Aura assuming 20% of $N = 50$ clients are corrupted. Note that the number of attackers included in the global update varies even without defense due to random client selection.

the vector, denoted by N_Z and and N_O respectively, followed by multiplying them with the square of the respective scales and adding the results, i.e. $N_Z \cdot (\mathsf{s}_Y^{min})^2 + N_O \cdot (\mathsf{s}_Y^{max})^2$. Furthermore, computing the number of ones N_O corresponds to the bit-aggregation of the vector $\vec{Y}$, for which our aggregation methods discussed in §III-A can be utilized.

Algorithm 4 Cosine Distance Calculation

```
procedure COSINE(($\vec{\sigma}_Y, \mathsf{s}_Y^{min}, \mathsf{s}_Y^{max}$), $\vec{S}$)
    $\mathsf{L}_2^Y \leftarrow$ L2-NORMQ($\vec{\sigma}_Y, \mathsf{s}_Y^{min}, \mathsf{s}_Y^{max}$)
    $\mathsf{L}_2^S \leftarrow \|\vec{S}\|$ // Computes L$_2$-norm
    $\alpha \leftarrow$ SUM($\vec{S}$) // Sum of elements of $\vec{S}$
    $\beta \leftarrow$ INNER-PRODUCT($\vec{\sigma}_Y, \vec{S}$)
    $\gamma = \mathsf{s}_Y^{min} \cdot \alpha + \beta \cdot (\mathsf{s}_Y^{max} - \mathsf{s}_Y^{min})$
    return $\gamma / (\mathsf{L}_2^Y \cdot \mathsf{L}_2^S)$
end procedure
```

Alg. 4 is used to compute the cosine distance between a quantized vector $\vec{Y}_\sigma$ and a reference vector $\vec{S}$. The cosine distance is given by $\frac{\vec{Y}_\sigma \odot \vec{S}}{\|\vec{Y}_\sigma\| \cdot \|\vec{S}\|}$, where $\|\cdot\|$ corresponds to the L$_2$-norm of the input vector. Using Eq. 4, we can write

$$\begin{aligned}\vec{Y}_\sigma \odot \vec{S} &= (\mathsf{s}_Y^{min} \oplus \vec{\sigma}_Y \circ (\mathsf{s}_Y^{max} - \mathsf{s}_Y^{min})) \odot \vec{S} \\ &= \mathsf{s}_Y^{min} \odot \vec{S} \ + \ (\vec{\sigma}_Y \odot \vec{S}) \cdot (\mathsf{s}_Y^{max} - \mathsf{s}_Y^{min}).\end{aligned}$$

Thus, the inner product computation of $\vec{Y}_\sigma \odot \vec{S}$ reduces to computing $\vec{\sigma}_Y \odot \vec{S}$, followed by two multiplications.

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

Similarly as for ResNet9 (cf. Fig. 12), 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 10% or more when using no compression. One third of the training rounds are impacted by about 10% 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 more robust against untargeted poisoning.

With ScionFL-Aura, the accuracy reduction is still 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.