Armadillo: Robust Secure Aggregation for Federated Learning with Input Validation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Secure aggregation protocols allow a server to compute the sum of inputs from a set of clients without learning anything beyond the sum (and what the sum implies). This paper introduces Armadillo, a single-server secure aggregation system for federated learning with input validation and robustness (guaranteed output delivery). Specifically, Armadillo allows the server to check if the input vectors satisfy some pre-defined constraints (e.g., the vectors have L_2, L_∞ norms bounded by a constant), and ensures the server can always obtain the sum of valid inputs.

Armadillo significantly improves the round complexity of ACORN-robust, a recent work by Bell et al. (USENIX Security '23) with similar security properties, from logarithmic rounds (to the number of clients) to constant rounds; concretely, when running one aggregation on 1K clients with corruption rate 10%, ACORN-robust requires at least 10 rounds while Armadillo has 3 rounds.

1 Introduction

Federated learning [52] is a mechanism to train models on private data distributed across many clients (e.g., mobile devices) under the orchestration of a central server, *without* having the server explicitly collect the data. It works by having the clients train local models using their own data and upload *only* their model weights to the server who aggregates the weights up (typically by averaging).

Under this distributed training mechanism, the clients never need to hand their private data to the server; however, prior works [53,68] in the machine learning community shows that the uploaded model weights of a client still leak information about the client's training data. Fortunately, the federated training only requires the server to learn the *sum* of the weights but not the individual weights. This motivates using secure aggregation to compute such sum, and indeed, many existing works [13,64,41,37,8,65,51] design protocols tailored for the federated learning setting, mostly aiming for efficient computation and communication.

A critical property that most of the prior protocols [13,8,64,41,51,48,65,37] lack is *robustness*: even if a single client misbehaves in the protocol execution, the server will possibly get a result that is vastly different from a correct sum, or even will not get any result (the protocol just aborts). Given the scale of the training participants, in practice, it is unlikely that every participating client is honest. Note that here the misbehaving is not the passive dropouts considered in prior work; it is actively deviating from the protocol prescription.

Beyond robustness, we want to aggregate only the valid client inputs (i.e., satisfy some pre-defined constraints). This is well motivated by adversarial machine learning: if the server incorporates malformed weights into the model, then the model accuracy may be downgraded, or even more severely, a backdoor could be injected into the model (altering the model's prediction on a minority of inputs while maintaining good overall accuracy on most inputs). Though such attacks are hard to provably prevent, previous work [58,9,24,50] offer criterion for input validation (e.g., bounds on L_2, L_∞ norms) that one can alleviate the effects of these attacks. Aside from the federated learning application, both robustness and input validation are also important for private statistics aggregation [14].

While a few existing works [50,9,24] delved into input validation, only ACORN-robust [9] provide robustness. ACORN-robust proposed a probabilistic approach to identify malicious clients and remove their inputs: when running a summation on n clients, the protocol requires $6 + O(\log n)$ rounds asymptotically; concretely, when running on 1K clients with corrupted rate 10% (20%), the protocol executes for at least 10 (15) rounds, except small probability. In this work, we propose a robust secure aggregation protocol with only 3 rounds. This achieves the same (or even smaller) round complexity compared to prior non-robust protocols [13,8,65,51]. Along the way, we also achieve a stronger property compared to ACORN-robust: the latter assumes a semi-honest server and we have malicious security. Next, we formally describe our problem, and our threat model and discuss the properties as mentioned above in detail.

1.1 Problem Statement and Thread Model

We proceed to formally describe our problem setting. A training process in federated learning consists of T iterations, running between the server and in total N clients. Each iteration has the same procedure: n clients (indexed by numbers from 1 to n) are selected from the N clients, where client i holds vector $\mathbf{x}_i$, and the goal is to let the server obtain the sum $\sum_{i=1}^{n} \mathbf{x}_i$ without revealing to the server anything except what can be implied by the sum.

In a real-world setting, a sum of all the n clients is hard to guarantee, as some clients may stop responding to the server during protocol execution (e.g., due to power failure or unstable connection). The server must continue without waiting for them to come back; otherwise, the training might be blocked for an unacceptable amount of time. Therefore, the goal (more precisely) is to compute the sum of the input vectors from the largest possible subset of the n clients.

Before we describe the desired properties, we first give the threat model and communication model of Armadillo.

Threat model. We follow the most commonly used model in federated learning literature [13,8,50,24,37,51,9], where there is a single (logical) server and n clients in each training iteration. We assume the adversary is static throughout an iteration, but it may change the corrupted set of clients across iterations, under the restriction that the corruption rate is always at most η.[1] We assume the server may also be corrupted. Within a complete iteration, we also assume at most δ fraction of n clients will drop out during the protocol execution.

Looking ahead, our protocol needs sub-sampling C out of n clients as a set $\mathcal{C}$ (to assist with the computation), so we introduce another notation $\eta_{\mathcal{C}}$ for the corruption rate of clients in $\mathcal{C}$. The relation between $n, C, \eta, \eta_{\mathcal{C}}$ is analyzed in Appendix H. Similarly, we assume at most $\delta_{\mathcal{C}}$ fraction of clients drop out when the server communicates with the clients in $\mathcal{C}$. See details in Section C.4 regarding the sub-sampling.

Communication model. Clients are heterogeneous devices with varying reliability (e.g., cellphones, laptops) and can stop responding due to device or network failure. We assume there is an implicit distribution for client response time.

Each client communicates with the server through a private and authenticated channel. Private messages sent from clients to other clients are forwarded via the server and are encrypted with authenticated encryption under their shared symmetric keys (existing works [8,51] give ways to set up these keys with a public key infrastructure, or PKI). Public messages sent by a client to other clients are signed using the sender's public key (again, assuming a PKI) if the messages are the same for multiple recipients, otherwise, the client uses MAC under the symmetric key. We implicitly assume such client-to-client communication throughout our protocol description.

[1] This means the adversary cannot keep corrupting more and more users: for example, in practice, an adversary can corrupt users via distributing malware and the users will be refreshed (and uncorrupted) until the malware is detected.

Communication is performed in rounds, starting from the server. We will count a complete round trip (or *round*) as the communication from the server to clients and from clients back to the server. The server first sends out messages to clients, waits for a fixed amount of time to receive messages, and puts them in a message pool. When the waiting period is over, the server processes the messages in the pool and proceeds to the next round.

Concrete parameters. A recent survey of federated learning deployments [42] describes typical communication models and gives common parameters as outlined below. The size of the total population N is in the range of 100K–10M, wherein in a given iteration t, a set of 50–5K clients are chosen to participate. The number of training iterations T for a model is 500–10K. Input vectors ($\mathbf{x}_i$) have typically on the order of 1K–500K entries for the datasets we surveyed [46,45,20,19]. For malicious rate η, most of the prior work can handle η up to $1/3$, but in practical scenarios, it is much smaller [62] (e.g., 0.1%); the dropout rate δ depends on the waiting time set by the server and it is up to 10% in prior works [9,51] (if we allow more dropouts, the trained model will be biased towards the results from powerful devices).

1.2 Properties

Due to system and networking constraints in federated learning deployment, we aim for the aggregation protocol to ensure not just privacy but also additional properties, which we outline below. Formal definitions of these properties are given in Section D.

Privacy. Informally, an aggregation protocol is private if the server only learns the sum of inputs and what the sum implies. Formally, we can define privacy using an ideal/real simulation paradigm (see details in §D). Since privacy is well-studied in previous works, we will use most of this section to describe the next three properties.

Dropout resilience. This property is motivated by the instability of client devices and has been considered in many prior works [13,8,50,24,37,51,9]. Specifically, some clients may disconnect from the network during the aggregation process (which can be a passive or active failure) but we wish the protocol can still execute even if we drop those clients. We, therefore, require our private summation protocol to have *dropout resilience*: when all the parties follow the protocol, the server, in each iteration, will get a sum of inputs from the online clients (those who participate throughout this iteration).

Input validity. The decentralized feature of federated learning, on the negative side, allows clients to play adversarial attacks on the model by submitting maliciously generated weights (to inject backdoors to the model or downgrade the model accuracy). It is imperative for the server involved in the summation process to detect malformed inputs, which we call *input validity*.

Recent works in the machine learning community proposed effective criteria to classify valid weights (e.g., L_1, L_2 norms) [54,59,66,34]. If the client weights (input vector $\mathbf{x}_i$) are sent in the clear, it is easy for the server to apply these criteria to check the validity of the collected weights and exclude those invalid ones. However, checking input validity becomes a challenge in the private setting since the server does not know any individual weights. Furthermore, it's important to differentiate between the server's capability to identify malformed inputs and subsequently abort without a sum result (which already satisfies input validity) [50], and the ability to exclude malformed inputs and ultimately obtaining a valid sum. This latter capability aligns closely with the subsequent property we are about to describe.

Robustness. This property is motivated by the scale of users in federated learning: since the number of clients per iteration ranges from a few hundred to a few thousand, if the protocol aborts when clients misbehave, the cost for the server to re-run the protocol is prohibitively high. We, therefore, require guaranteed output delivery, which we call *robustness*; namely, if the server is semi-honest, then it always obtains a sum of the inputs from the online honest clients even if malicious clients arbitrarily deviate from the protocol. Note that Armadillo does not guarantee robustness when the server acts maliciously—after all, in the federated learning application, the server is the one who wishes to receive the output.

	Client comm.	Client comp.	Server comm.	Server comp.	Rounds	Robust	Val.	Priv. agst. server
Effiel [24]	ℓn^2	ℓn^2	ℓn^3	ℓn^3	4	No*	Yes	Malicious
RoFL [50]	$\ell + \log n$	$\ell \log n$	$\ell n + n \log n$	$n\ell \log n$	6	No	Yes	Malicious
ACORN-detect [9]	$\ell + \log n$	$\ell + \log n$	$\ell n + n \log n$	ℓn	7	No	Yes	Malicious
ACORN-robust [9]	$\ell + \log^2 n$	$\ell \log n + \log^2 n$	$\ell n + n \log^2 n$	$\ell n + n \log^2 n$	$6 + \log n$	Yes	Yes	Semi-honest
Flamingo [51]	Regular: $\ell + C$ Helper*: $n + C$	Regular: $\ell + C$ Helper*: $n + C$	$\ell n + Cn$	$\ell n + Cn$	3	No	No	Malicious
Armadillo (this work)	Regular: $\ell + C$ Helper: $n + C$	Regular: $\ell + C$ Helper: $n + C$	$\ell n + Cn$	$\ell n + Cn$	3	Yes	Yes	Malicious

Figure 1: Asymptotic communication and computation cost for one training iteration, where vector length is ℓ and number of clients per iteration is n; for simplicity, we omit the asymptotic notation $O(\cdot)$ in the table. In practice we have $n < \ell$ (§1.1). All the costs include zero-knowledge proof for the protocols with input validation. The round complexity excludes any setup that is one-time. The round complexity of ACORN-detect are counted using the fixed version (Appendix H.3). The header "Priv. agst. server" means if the protocol achieves privacy against a semi-honest or malicious server. We choose the baseline protocols that has similar properties as ours or use the similar model as ours: ACORN-detect, Eiffel and RoFL have input validation, and ACORN-robust has robustness. Flamingo also uses the idea of sub-sampling helpers (which they call decryptors); we denote the number of sampled clients as C where $C = o(n)$. In Flamingo, the helper has asymptotic cost slightly larger than n when dropouts happen (marked $*$ in the table). Eiffel can additionally have robustness with expensive replication which we do not include it here (marked $*$ in the table).

1.3 Our Contributions

We introduce Armadillo, a secure aggregation protocol that achieves all the outlined properties: it has dropout resilience and ensures privacy even when the server and a subset of clients act maliciously (as detailed in the threat model presented in §1.1). The server in our protocol can verify if the client inputs satisfy certain norm constraints, and the protocol is robust against malicious clients.[2]

Figure 1 offers an in-depth comparison between our protocol and prior works in terms of asymptotic costs (computation, communication, and rounds). Our contributions can be summarized as:

- Armadillo reduces the asymptotic round complexity from $6 + O(\log n)$ of ACORN-robust protocol [9] to 3 rounds, while keeping the asymptotic computation and communication cost on par with ACORN-robust, assuming $C = O(\log^2 n)$. See Figure 1 for details.
- For concrete performance, Armadillo's client computation is roughly $1.5\times$ smaller than that of ACORN (Fig.2a,2b). Importantly, we have 3–$7\times$ improvement for round complexity concretely (Fig.3a), which translates to $10\times$ improvement for performing a complete sum (Fig.4). Our competitive advantage of round complexity over ACORN-robust is bigger when there are more clients (n is larger) or the malicious rate is higher (η is larger).
- Our protocol has privacy against a malicious server, which is an improvement from the prior robust protocol of [9] (ACORN-robust's threat model assumes a semi-honest server). We also address a mild security concern in ACORN-family protocols (Appendix H.3); the fix incurs an additional round to their original protocol.

Due to space constraints, we defer related work to Section B.

2 Preliminaries

Notation. Let $[z]$ denote the set $\{1, 2, \ldots, z\}$. We use $[a, b]$ to denote the set $\{x \in \mathbb{N} : a \leq x \leq b\}$. We use bold lowercase letters (e.g. $\mathbf{u}$) to denote vectors and bold upper case letters (e.g., $\mathbf{A}$) to denote matrices. Unless specified, vectors are column vectors. For distribution $\mathcal{D}$, we use $a \leftarrow \mathcal{D}$ to denote sampling a from $\mathcal{D}$. For a vector $\mathbf{v}$, we use $\lfloor \mathbf{v} \rceil_c$ to denote rounding each entry of $\mathbf{v}$ to nearest multiples of c. For two vectors $\mathbf{v}_1$ of length ℓ_1, $\mathbf{v}_2$ of length ℓ_2, we use $\mathbf{v}_1 | \mathbf{v}_2$ to denote the

[2]We do not consider robustness when the server is malicious, because in the federated learning setting, the server wants to get the aggregation result.

concatenation of them which is a vector of length $\ell_1 + \ell_2$. We use $\|\mathbf{v}\|_2$ to denote L_2 norm of $\mathbf{v}$ and use $\|\mathbf{v}\|_\infty$ to denote the largest entry in $\mathbf{v}$.

We defer our cryptographic preliminaries to Section A

3 Technical Overview

In this section, we describe our construction for computing one sum. We discuss computing multiple sums and related security issues in Section C.4.

3.1 A two-layer secure aggregation

We start with a base secure aggregation scheme with only dropout resilience and semi-honest security. The high-level idea is to reduce the secure aggregation for long vectors to secure aggregation for short vectors, utilizing the key and message homomorphism of Regev's encryption. Note that a similar idea has appeared in many similar or orthogonal settings [65,6,48,16,35,10] but none of these works addresses the robustness.

Each client $i \in [n]$ holding an input vector $\mathbf{x}_i$ (model weights) samples a Regev's encryption key $\mathbf{k}_i$ and sends to the server $\mathbf{y}_i = \mathsf{Enc}(\mathbf{k}_i, \mathbf{x}_i)$; note that $\mathbf{y}_i$ is of the same length as input $\mathbf{x}_i$. Then the server simply computes the sum of all the $\mathbf{y}_i$'s. Note that

$$\mathbf{y} := \sum_{i\in[n]} \mathbf{y}_i = \sum_{i\in[n]} \mathsf{Enc}(\mathbf{k}_i, \mathbf{x}_i) = \mathsf{Enc}(\sum_{i\in[n]} \mathbf{k}_i, \sum_{i\in[n]} \mathbf{x}_i).$$

To get the sum result $\sum_{i\in[n]} \mathbf{x}_i$, the server just needs to know $\mathbf{k} := \sum_{i=1}^n \mathbf{k}_i$ to decrypt $\mathbf{y}$. For this, we can use a black-box secure aggregation protocol to aggregate $\mathbf{k}_i$'s. Since the length of $\mathbf{k}_i$ is much shorter than the length of $\mathbf{x}_i$, we reduce an aggregation problem on long vectors $\mathbf{x}_i$'s to an aggregation problem on short vectors $\mathbf{k}_i$'s. Finally, the server computes $\mathsf{Dec}(\mathbf{k}, \mathbf{y})$, and the decryption succeeds if $\sum_{i\in[n]} \mathbf{e}_i < \frac{1}{2}\lfloor q/p \rfloor$. For simplicity, we call the aggregation for $\mathbf{k}_i$'s as *inner aggregation*, and the summation for $\mathbf{y}_i$'s as *outer aggregation*.

In this work, we instantiate the inner aggregation (that run on short vectors $\mathbf{k}_i$'s) as follows: we sub-sample[3] a small set of clients as helpers, and have each client i secret share $\mathbf{k}_i$ to C helpers using packed secret sharing in a threshold way, and we denote the shares of $\mathbf{k}_i$ as $s_i^{(1)}, \ldots, s_i^{(C)}$. These shares are sent under end-to-end encrypted channels (similarly as [13,8,51]) and happens *at the same time* when the client sends the ciphertext $\mathbf{y}_i$'s. Finally, each helper j locally adds up the received shares as $s^{(j)} = \sum_{i=1}^n s_i^{(j)}$ and then sends $s^{(j)}$ to the server who reconstructs $\mathbf{k}$ from $s^{(1)}, \ldots, s^{(C)}$.

The above inner-outer solution immediately handles dropouts, as opposed to the pairwise masking approach that some prior work [13,8,51] use which incurs extra rounds. If a client drops out when sending $\mathbf{y}_i$ and the shares, it will not affect the aggregation process at all (the server just safely ignores the client); if a helper client drops in the inner aggregation, later our protocol design and choice of parameters guarantee that as long as the active honest helpers is above the pre-set threshold, the server will always reconstruct the desired $\mathbf{k}$; so the inner-aggregation is robust to helper dropouts or malicious helpers who modify the shares.

The key challenge remaining is achieving robustness against malicious clients, which we discuss next.

3.2 Robustness

Recall the robustness property we briefly described in Section 1.1: the server will always get a sum of the inputs from honest clients; namely, once the clients send to the server the encryption of $\mathbf{x}_i$'s, no malicious client should be able to change the sum of those $\mathbf{x}_i$'s anymore. To this end, we require that

1. In the outer aggregation, each client encrypts input vector $\mathbf{x}_i$ using key $\mathbf{k}_i$ correctly.
2. The $\mathbf{k}_i$ in the inner aggregation is consistent with what was used in the outer aggregation.

[3] We discuss how to do sub-sampling securely in Section C.4.

3. In the inner aggregation, each client secret-shares $\mathbf{k}_i$ using a polynomial of the degree as prescribed (this is to ensure the inner aggregation itself is robust).

We express *all* these requirements using only simple *inner-product relations*. As a result, our robust protocol at a high level works as follows: each client sends to the server the commitments to its input, its key, and the shares of the key[4]; and then proves that the above requirements hold under the commitments. Crucially, these requirements are simply a few inner-product statements. In short, a client in our robust protocol will just send the commitments along with a few inner-product proofs in addition to what was sent in the base protocol (ciphertext $\mathbf{y}_i$ and the Shamir shares of $\mathbf{k}_i$).

For simplicity, we start by describing the third bullet above and then describe the second bullet. We will discuss the first bullet in the next Section. Due to space constraints, we defer details on the various proof techniques to Section C

Theorem 1 (Cost of proofs in Armadillo)**.** Given a set of parameters (λ, ℓ, q, C). Let $\mathbf{k} \in \mathbb{Z}_q^\lambda, \mathbf{s} \in \mathbb{Z}_q^C, \mathbf{M} \in \mathbb{Z}_q^{\lambda\times C}, \mathbf{x} \in \mathbb{Z}_q^\ell$. Let $\mathbb{G}$ be a group of size q. Let Δ be a constant and $\mathbf{w}$ be a constant vector. Let

$$
\begin{aligned}
&\mathbb{CS}_{\text{Shamir}} : \{\mathsf{io} : \mathsf{com}(\mathbf{s}),\ \mathsf{st} : \langle \mathbf{w}, \mathbf{s}\rangle = 0,\ \mathsf{wt} : (\mathbf{s}, \mathbf{k})\}.\\
&\mathbb{CS}_{\text{bind}} : \{\mathsf{io} : (\mathsf{com}(\mathbf{s}), \mathsf{com}(\mathbf{k})),\ \mathsf{st} : \mathbf{k} = \mathbf{M}\cdot\mathbf{s},\ \mathsf{wt} : (\mathbf{s}, \mathbf{k})\}.\\
&\mathbb{CS}_{\text{enc}} : \{\mathsf{io} : (\mathsf{com}(\mathbf{k}), \mathsf{com}(\mathbf{x}), \mathsf{com}(\mathbf{e})),\\
&\qquad\qquad \mathsf{st} : \mathbf{y} = \mathbf{A}\cdot\mathbf{k} + \mathbf{e} + \Delta\cdot\mathbf{x},\\
&\qquad\qquad \|\mathbf{e}\|_2 < B_{\mathbf{e}}(L_2), \|\mathbf{x}\|_\infty < B_{\mathbf{x}}(L_\infty), \|\mathbf{x}\|_2 < B_{\mathbf{x}}(L_2),\\
&\qquad\qquad \mathsf{wt} : (\mathbf{k}, \mathbf{x}, \mathbf{e})\}.
\end{aligned}
$$

There exist commit-and-proof protocols (based on group $\mathbb{G}$) for proving the above statements with the following cost, dominated by the inner-product proof (IP) invocations:

- 2 IPs of length 4ℓ,
- 4 IP of length ℓ,
- 1 IP of length λ,
- 1 IP of length C,
- 1 IP of length $\lambda + C$,

where we omit the lower order terms and write e.g., $\ell + 256$, as ℓ.

We defer additional discussions on our construction to the appendix. See Section C.4.

4 Implementation and Evaluation

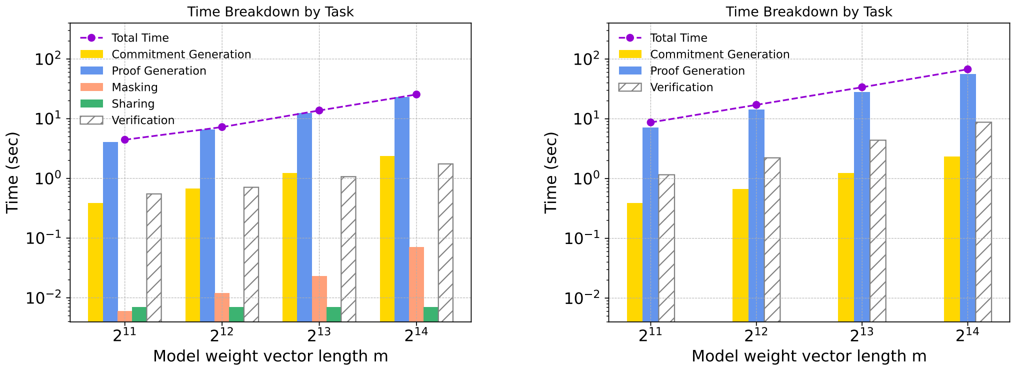
In this section, we provide benchmarks to answer the following questions:

- What are the concrete costs of the client and the server, for aggregation and proofs, respectively?
- What is the cost of the helpers and how does it compare to the cost of regular clients?
- How is Armadillo's performance compared to prior works with similar properties, i.e., ACORN-robust?

To better understand the concrete cost, readers can find the cost overview of the client and the server in Section F

Experimental results. Figure 2a shows the computation time for a client by breaking down to several parts: "Commitment generation" is the time for the client to commit to all the vectors and the secret shares required in the proof, "Proof generation" is the time to create the proof (using Nova), and "Masking" is the time for the client to compute the masked input vector (with the preprocessing optimization in Section E.2), and "Sharing" is the time for generating Shamir shares for the helpers. Note that all these costs are independent of the number of clients, as long as the number of helpers C is fixed. We also depict the time for verifying a single client's proof in Figure 2a (to contrast with the client costs), but this is done by the server. Our protocol has the property that the server cost scales linearly with the number of clients.

[4]For our construction, each of the vector components and the shares are committed using different generators.



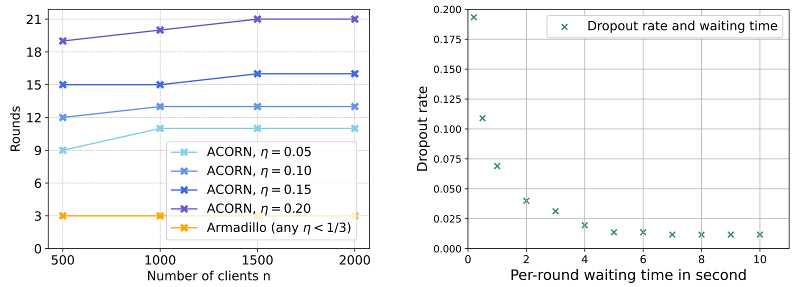
(a) Armadillo. (b) ACORN-robust.

Figure 2: Computational time per client in Armadillo and ACORN depicted as log scale, for different input vector lengths. For Armadillo, the total time per client includes commitment generation, proof generation, masking and sharing; the per-client cost is independent of the number of clients. The verification time depicted is for per client proof and the verification is done by the server. For ACORN, the total time per client we show includes the commitment generation and proof generation. We do not include the computational cost of the cheating client identification, for which the computation cost is negligible compared to the proof generation (see details in Algorithm 4 in [9] and our description in §4).

For ACORN-robust, since they did not have implementation and benchmarks, we depict in Figure 2b the time of the dominating computation of their protocol. Specifically, ACORN-robust works by first doing aggregation and then identifying and removing the invalid inputs. The bulk of computation for clients happens in the aggregation phase, and during the identification phase, the client only provides the server with the messages it stored from previous rounds, without extra computation. See Algorithm 4 in [9] for details. Therefore, we can focus on the aggregation-phase client computation, where the dominating cost is generating commitments and creating proofs. We implemented and microbenchmarked their commitment and proof generation (for L_2, L_∞ norms on inputs), instantiating their inner-product proofs with Bulletproof [17,28] as reported in their paper. In sum, what depicted in Figure 2b will be a slight underestimate of their client cost.

From Figure 2a and 2b we can see that the client computation of Armadillo is ~1.5$\times$ better than that of ACORN-robust. However, what makes a big difference is the round complexity. In Figure 3a, we depict the round complexity for ACORN-robust under different settings of n and η, based on their probabilistic analysis (Theorem 4.1, [9]). Since their protocol does not have a fixed number of rounds (their identification protocol runs in a probabilistic iterative manner), we count the number of rounds such that ACORN protocol ends with more than 0.9 probability. Our protocol remains the same number of rounds (3) in all the settings we show. In the best setting when $n = 500$ and $\eta = 0.05$, ACORN-robust still has 9 rounds (3$\times$ of ours); and in the worst setting when $n = 2000$ and $\eta = 0.2$, their protocol has 21 rounds (7$\times$ of ours). Also, in these rounds, the ACORN server communicates with all the clients; while in Armadillo the server communicates with all the clients in the first round, and in the rest of the 2 rounds the server communicates with only the helpers.

Figure 4 shows how the round complexity translates to the run time of a complete summation. To do this, we first run a network simulator ABIDES [18] to get the relation between the dropouts and the server waiting time (Fig.3b). If we fix a message arrival distribution, then the shorter the time that the server waits, the less number of messages it will get. If a protocol only tolerates dropout, say 5% (meaning that if 10% of the clients drop then the protocol is insecure), then it means that the server needs to wait until 95% messages arrive. So if the protocol tolerate less dropout, say 1%, then the server needs to wait until 99% messages to arrive, which takes longer than the former case. In the extreme case, if the server needs to wait for 100% of the messages to arrive, then the protocol could never terminate because there could be a client that goes offline in the middle of the execution and stays offline forever.



(a) Number of rounds for ACORN-robust and Armadillo under different settings of n and η.

(b) The server waiting time determined by target dropout rates, fixing a message arrival distribution.

Figure 3: The number of rounds for ACORN-robust and Armadillo under different η, and the server waiting time under different target δ. These two sets of information is useful for estimating the total round trip time in Figure 4.

	η	δ	#rounds	avg δ per round	per-round waiting (second)	total round trip time (second)
Armadillo	0.1	0.1	3	0.0333	4	12
	0.2	0.1	3	0.0333	4	12
	0.1	0.2	3	0.0667	2	6
ACORN	0.1	0.1	12	0.0083	10	120
	0.2	0.1	19	0.0053	10	190
	0.1	0.2	12	0.0167	10	120

Figure 4: Estimated total time spent on round trips (a server and 500 clients). Fixing a set of δ and η (which should be set within the bound that the protocol can tolerate), we can calculate the average dropouts per round that a protocol can tolerate (dividing total dropout δ by the number of rounds). Then fixing a per-round dropout, we determine server waiting time using the data points in Figure 3b. The total round trip time is estimated as the waiting time per round (Fig.3b) multiplied with the number of rounds (Fig.3a).

In short, fixing the dropout rate that a protocol can tolerate, then there will be a big difference in the server waiting time when the protocol has 12 rounds vs. the protocol has 3 rounds. Figure 4 explains how we estimate the total round trip time.

5 Conclusion

In this work, we present Armadillo which focuses on achieving robustness by detecting and removing cliens behaving maliciously. Armadillo outperforms the state-of-the-art ACORN protocol [9], as backed by our benchmarking efforts. We point out the following limitations of the work:

- It is known that the Regev encryption scheme can be made more efficient by relying on the Ring-LWE assumption. This work does not explore this counterpart, which is a direction for future research.
- While identifying malicious behavior can lead to robustness, there has been independent lines of work such as RSA [49] that achieves robustness in the face of byzantine action, without relying on identifying malicious behavior. This work does not investigate composing results with these alternate mechanism for robustness.
- While secure aggregation has the capability to aggregate model updates without any loss in accuracy and privacy, we leave it as future work to use Armadillo for end-to-end model training.

Secure aggregation for training iteration t

Server and clients agree on public parameters: LWE parameters $(\lambda, m, p, q, \mathbf{A} \in \mathbb{Z}_q^{\lambda \times m})$, the group $\mathbb{G}$ (of order q) for the commit-and-proof system, the norm bound $B_{\mathbf{x}}(L_\infty), B_{\mathbf{x}}(L_2), B_{\mathbf{e}})$. Let $\Delta = \lfloor q/p \rfloor$. The dropout rate is δ and malicious rate over n clients is η across all rounds in each iteration. The set of C helpers is determined via a random beacon or Feige election (Section C.4), with threshold being d.

Round 1 (Server → Clients)
Server notifies n clients (indexed by numbers in $[n]$) to start iteration $t \in [T]$.

Round 1 (Clients → Server)
Client $i \in [n]$ on input $\mathbf{x}_i \in \mathbb{Z}_q^m$, computes the following:

1. $\mathbf{k}_i \xleftarrow{\$} \mathbb{Z}_q^\lambda$, $\mathbf{e}_i \leftarrow \chi^m$,
2. $\mathbf{y}_i = \mathbf{A} \cdot \mathbf{k}_i + \mathbf{e}_i + \Delta \mathbf{x}_i$, where $\mathbf{y}_i \in \mathbb{Z}_q^m$,
3. Compute degree-d packed secret sharing of $\mathbf{k}_i$ as $\mathbf{s}_i = (s_i^{(1)}, \dots, s_i^{(C)})$.
4. Computes commitment to vector $\mathbf{k}_i$ as $\mathsf{com}(\mathbf{k}_i)$ and to vector $\mathbf{x}_i$ as $\mathsf{com}(\mathbf{x}_i)$, // two elements in $\mathbb{G}$
5. Computes commitment to shares $s_i^{(j)}$ as $\mathsf{com}_{G_j}(s_i^{(j)})$ for $j \in [C]$, where $\{G_j\}_{j \in [C]}$ are a set of generators in $\mathbb{G}$. // C elements in $\mathbb{G}$
6. Set constraint system $\{\mathsf{io} : (\mathsf{com}(\mathbf{s}_i), \mathbf{w}), \ \mathsf{st} : \langle \mathbf{s}_i, \mathbf{w} \rangle = 0, \ \mathsf{wt} : \mathbf{s}_i\}$, and computes $\pi_{\text{Shamir}} \leftarrow \Pi_{\text{ip}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$, where $\mathbf{w}$ is computed as: $m^*(X) \leftarrow_\$ \mathbb{F}[X]_{\leq C-d-2}$ and let $\mathbf{w} := (v_1 \cdot m^*(1), \dots, v_n \cdot m^*(C))$.
7. Set constraint system $\{\mathsf{io} : (\mathsf{com}(\mathbf{s}_i), \mathsf{com}(\mathbf{k}_i), \mathbf{M}), \ \mathsf{st} : \mathbf{k}_i = \mathbf{M} \cdot \mathbf{s}_i, \ \mathsf{wt} : (\mathbf{s}_i, \mathbf{k}_i)\}$, and compute $\pi_{\text{bind}} \leftarrow \Pi_{\text{linear}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$, // $O(\log C + \log \lambda)$ elements in $\mathbb{G}$, see algorithm in Figure 6
8. Set constraint system $\mathbb{CS}_{\text{enc}}$: $\{\mathsf{io} : (\mathsf{com}(\mathbf{k}), \mathsf{com}(\mathbf{x}), \mathsf{com}(\mathbf{e})),$ $\mathsf{st} : \mathbf{y} = \mathbf{A} \cdot \mathbf{k} + \mathbf{e} + \Delta \cdot \mathbf{x}, \|\mathbf{e}\|_2 < B_{\mathbf{e}}(L_2), \|\mathbf{x}\|_\infty < B_{\mathbf{x}}(L_\infty), \|\mathbf{x}\|_2 < B_{\mathbf{x}}(L_2), \mathsf{wt} : (\mathbf{k}, \mathbf{x}, \mathbf{e})\}$, and compute $\pi_{\text{enc}} \leftarrow \Pi_{\text{enc}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$. // $O(\log m)$ elements in $\mathbb{G}$, see algorithm in Section 3

Client $i \in [n]$ sends a tuple to the server:

$\{$"server" : $(\mathbf{y}_i, \mathsf{com}(\mathbf{k}_i), \mathsf{com}(\mathbf{x}_i), \mathsf{com}(\mathbf{e}_i), \{\mathsf{com}_{G_j}(s_i^{(j)})\}_{j \in [C]},$
$\pi_{\text{Shamir}}, \pi_{\text{bind}}, \pi_{\text{input}}, \pi_{\text{enc}})$;
"helper $j \in \mathcal{C}$" : $(s_i^{(j)}, r_i^{(j)})\}$

where $r_i^{(j)}$ is the randomness for $\mathsf{com}_{G_j}(s_i^{(j)})$.

// Note that every message from clients intended for clients/helpers is symmetrically encrypted.

Round 2 (Server → Helpers)
Let $\mathcal{S}_1 \subset [n]$ be the clients who sent the prescribed messages in Round 1.
The server does the following computation for each client $i \in \mathcal{S}_1$:

1. Compute $\mathsf{com}(\mathbf{s}_i) := \prod_{j \in [C]} \mathsf{com}_{G_j}(s_i^{(j)})$,
2. Run $\Pi_{\text{ip}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi_{\text{Shamir}})$, $\Pi_{\text{linear}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi_{\text{bind}})$, $\Pi_{\text{enc}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi_{\text{enc}})$.
3. If all the proofs are valid, then send to each helper j the share commitment $\mathsf{com}_{G_i}(s_i^{(j)})$.

Round 2 (Helpers → Server) Each helper $j \in [C]$:

1. If received less than $(1 - \delta - \eta)n$ shares $s_i^{(j)}$, abort. Otherwise, it verifies $(s_i^{(j)}, r_i^{(j)})$ against the commitment $\mathsf{com}_{G_j}(s_i^{(j)})$, denote the set of clients whose commitments are valid as $\mathcal{S}_2$.
2. Sign the set $\mathcal{S}_2$ and sends the signature to all the other helpers via the server.

Round 3 (Server → Helpers) Server forwards the signatures to helpers.

Round 3 (Helpers → Server) Each helper $j \in [C]$: if received more than $2C/3$ valid signatures on the same set (including its own signature), then continues. Otherwise abort. Computes $s^{(j)} := \sum_{i \in \mathcal{S}_2} s_i^{(j)}$ and sends it to the server.
Server reconstruct the shares $\{s^{(j)}\}_{j \in C}$ to $\mathbf{k}$, and computes $\mathbf{y} := \sum_{i \in \mathcal{S}_3} \mathbf{y}_i$.
Server computes $\lfloor \mathbf{y} - \mathbf{A} \cdot \mathbf{k} \mod q \rceil_\Delta$.

Figure 5: A secure aggregation protocol with dropout resilience, robustness, and input validity.

References

[1] Cloudflare randomness beacon. https://developers.cloudflare.com/randomness-beacon/.

[2] I. T. Abhiram Kothapalli, Srinath Setty. Nova: Recursive zero-knowledge arguments from folding schemes. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2022.

[3] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. In *Journal of Mathematical Cryptology*, 2015.

[4] B. Alon, M. Naor, E. Omri, and U. Stemmer. MPC for tech giants (GMPC): Enabling gulliver and the lilliputians to cooperate amicably, 2022. https://eprint.iacr.org/2022/902.

[5] S. Angel, A. J. Blumberg, E. Ioannidis, and J. Woods. Efficient representation of numerical optimization problems for SNARKs. In *Proceedings of the USENIX Security Symposium*, 2022.

[6] M. Ball, H. Li, H. Lin, and T. Liu. New ways to garble arithmetic circuits. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2023.

[7] L. Bangalore, M. H. F. Sereshgi, C. Hazay, and M. Venkitasubramaniam. Flag: A framework for lightweight robust secure aggregation. In *Flag: A Framework for Lightweight Robust Secure Aggregation*, 2023.

[8] J. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2020.

[9] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun. ACORN: input validation for secure aggregation. In J. A. Calandrino and C. Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 4805–4822. USENIX Association, 2023.

[10] J. Bell-Clark, A. Gascón, B. Li, M. Raykova, and P. Schoppmann. Willow: Secure aggregation with one-shot clients. Cryptology ePrint Archive, Paper 2024/936, 2024.

[11] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1988.

[12] F. Benhamouda, A. Degwekar, Y. Ishai, and T. Rabin. On the local leakage resilience of linear secret sharing schemes. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2018.

[13] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[14] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai. Lightweight techniques for private heavy hitters. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

[15] J. Bootle. Efficient multi-exponentiation. https://jbootle.github.io/Misc/pippenger.pdf.

[16] J. Brorsson and M. Gunnarsson. DIPSAUCE: Efficient private stream aggregation without trusted parties, 2023. https://eprint.iacr.org/2023/214.pdf.

[17] B. Bunz, J. Bootle, P. W. Dan Boneh, Andrew Poelstra, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[18] D. Byrd, M. Hybinette, and T. H. Balch. ABIDES: Agent-based interactive discrete event simulation environment. https://github.com/abides-sim/abides, 2020.

[19] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar. Leaf: A benchmark for federated settings. https://github.com/TalwalkarLab/leaf.

[20] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018. https://arxiv.org/abs/1812.01097.

[21] I. Cascudo and B. David. SCRAPE: Scalable randomness attested by public entities. In D. Gollmann, A. Miyaji, and H. Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 537–556, Kanazawa, Japan, July 10–12, 2017. Springer, Heidelberg, Germany.

[22] H. Chen, L. Chua, K. Lauter, and Y. Song. On the concrete security of LWE with small secret, 2020. https://eprint.iacr.org/2020/539.pdf.

[23] J. H. Cheon, D. Kim, D. Kim, J. Lee, J. Shin, and Y. Song. Lattice-based secure biometric authentication for hamming distance. In *Information Security and Privacy*, 2021.

[24] A. R. Chowdhury, C. Guo, S. Jha, and L. van der Maaten. Eiffel: Ensuring integrity for federated learning. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2022.

[25] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[26] D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi. Lwe with side information: Attacks and concrete security estimation. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2020.

[27] S. Das, V. Krishnan, I. M. Isaac, and L. Ren. Spurt: Scalable distributed randomness beacon with transparent setup. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

[28] H. de Valence, C. Yun, and O. Andreev. Bulletproof implementation based on delak-cryptography. https://github.com/dalek-cryptography/bulletproofs.

[29] U. Feige. Noncryptographic selection protocols. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.

[30] F. A. Feldman. Fast spectral tests for measuring nonrandomness and the DES. In C. Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 243–254, Santa Barbara, CA, USA, Aug. 16–20, 1988. Springer, Heidelberg, Germany.

[31] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1987.

[32] B. Feng. Multi-scalar multiplication (MSM). https://hackmd.io/@tazAymRSQCGXTUKkbh1BAg/Sk27liTW9.

[33] M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1992.

[34] C. Fung, C. J. M. Yoon, and I. Beschastnikh. The limitations of federated learning in sybil settings. In *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2020.

[35] A. Gascón, Y. Ishai, M. Kelkar, B. Li, Y. Ma, and M. Raykova. Computationally secure aggregation and private information retrieval in the shuffle model. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2024.

[36] C. Gentry, S. Halevi, and V. Lyubashevsky. Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2022.

[37] Y. Guo, A. Polychroniadou, E. Shi, D. Byrd, and T. Balch. Microfedml: Privacy preserving federated learning for small weights. Cryptology ePrint Archive, Paper 2022/714, 2022. https://eprint.iacr.org/2022/714.

[38] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *Proceedings of the USENIX Security Symposium*, 2023.

[39] A. Jain, H. Lin, and S. Saha. A systematic study of sparse lwe. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2024.

[40] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 1984.

[41] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramchandran. FastSecAgg: Scalable secure aggregation for privacy-preserving federated learning. In *ICML Workshop on Federated Learning for User Privacy and Data Confidentiality*, 2020.

[42] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawit, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, H. Eichner, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. Theertha Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao. Advances and open problems in federated learning. In *Foundations and Trends in Machine Learning*, 2021.

[43] D. Kim, D. Lee, J. Seo, and Y. Song. Toward practical lattice-based proof of knowledge from hint-MLWE. In H. Handschuh and A. Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 549–580, Santa Barbara, CA, USA, Aug. 20–24, 2023. Springer, Heidelberg, Germany.

[44] A. Kothapalli and S. Setty. HyperNova: Recursive arguments for customizable constraint systems. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2024.

[45] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[46] A. Krizhevsky, V. Nair, and G. Hinton. The CIFAR-100 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[47] J. Lee, D. Kim, D. Kim, Y. Song, J. Shin, and J. H. Cheon. Instant privacy-preserving biometric authentication for hamming distance, 2018. https://eprint.iacr.org/2018/1214.

[48] H. Li, H. Lin, A. Polychroniadou, and S. Tessaro. Lerna: Secure single-server aggregation via key-homomorphic masking. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2023.

[49] 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 *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[50] H. Lycklama, L. Burkhalter, A. Viand, N. Kuchler, and A. Hithnawi. RoFL: Robustness of secure federated learning. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2023.

[51] Y. Ma, J. Woods, S. Angel, A. Polychroniadou, and T. Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2023.

[52] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the Artificial Intelligence and Statistics Conference (AISTATS)*, 2017.

[53] L. Melis, G. Danezis, and E. D. Cristofaro. Efficient private statistics with succinct sketches. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[54] T. D. Nguyen, P. Rieger, H. Chen, H. Yalame, H. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, S. Zeitouni, F. Koushanfar, A.-R. Sadeghi, and T. Schneider. FLAME: Taming backdoors in federated learning. In *Proceedings of the USENIX Security Symposium*, 2022.

[55] D. Pasquini, D. Francati, and G. Ateniese. Eluding secure aggregation in federated learning via model inconsistency. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2022.

[56] N. Pippenger. On the evaluation of powers and monomials. *SIAM Journal of Computation*, 1980.

[57] M. O. Rabin and J. O. Shallit. Randomized algorithms in number theory. *Communications on pure and applied mathematics*, 1986.

[58] M. Rathee, C. Shen, S. Wagh, and R. A. Popa. Elsa: Secure aggregation for federated learning with malicious actors. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2023.

[59] P. Rieger, T. D. Nguyen, M. Miettinen, and A. Sadeghi. Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2022.

[60] S. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2020.

[61] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[62] 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 *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2022.

[63] J. So, R. E. Ali, B. Guler, J. Jiao, and A. S. Avestimehr. Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning. In *The Association for the Advancement of Artificial Intelligence (AAAI)*, 2023.

[64] J. So, C. He, C.-S. Yang, S. Li, Q. Yu, R. E. Ali, B. Guler, and S. Avestimehr. LightSecAgg: a lightweight and versatile design for secure aggregation in federated learning. In *Proceedings of Machine Learning and Systems*, 2022.

[65] T. Stevens, C. Skalka, C. Vincent, J. Ring, S. Clark, and J. Near. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. In *Proceedings of the USENIX Security Symposium*, 2022.

[66] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong. Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients. In A. Zhang and H. Rangwala, editors, *ACM SIGKDD conference on Knowledge Discovery and Data Mining (KDD)*, 2022.

[67] Y. Zhao, H. Zhou, and Z. Wan. Superfl: Privacy-preserving federated learning with efficiency and robustness. Cryptology ePrint Archive, Paper 2024/081, 2024. `https://eprint.iacr.org/2024/081`.

[68] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. In *Neural Information Processing Systems (NeurIPS)*, 2019.

A Cryptographic Preliminaries

Our construction utilizes two properties of Regev's encryption: key homomorphism and message homomorphism. We give the details below.

Regev's encryption. Given a secret key $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^\lambda$, the encryption of a vector $\mathbf{x} \in \mathbb{Z}_p^m$ is

$$(\mathbf{A}, \mathbf{c}) := (\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e} + \lfloor q/p \rfloor \cdot \mathbf{x}),$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times \lambda}$ is a random matrix ($m > \lambda$), and $\mathbf{e} \xleftarrow{\$} \chi^m$ is an error vector, χ is a discrete Gaussian distribution. Decryption is computed as $(\mathbf{c} - \mathbf{A}\mathbf{s}) \bmod q$ and rounding each entry to the nearest multiples of $\lfloor q/p \rfloor$. The decrypted result is only correct if entries in $\mathbf{e}$ are less than $\frac{1}{2} \cdot \lfloor q/p \rfloor$.

Looking ahead, for efficiency reasons, we are interested in small λ (e.g., 40–100) and entries of $\mathbf{s}$ being small, and in Section E, we give concrete parameter selection according to recent security analysis on LWE [3,22,26].

As observed in a few works in orthogonal areas [38], Regev's encryption remains secure even if $\mathbf{A}$ is made public and the same matrix $\mathbf{A}$ is used to encrypt polynomially many messages, as long as the secret key $\mathbf{s}$ and the noise $\mathbf{e}$ are independently chosen in each instance of encryption. In our case, $\mathbf{A}$ is a public random matrix and it can be generated by a trusted setup (i.e., random beacon service [27,1] generates a seed and the parties use PRG to expand the seed to matrix $\mathbf{A}$). Since $\mathbf{A}$ can be reused, so this setup only needs to run once.

Now, given two ciphertexts $(\mathbf{A}, \mathbf{c}_1), (\mathbf{A}, \mathbf{c}_2)$ of vectors $\mathbf{x}_1, \mathbf{x}_2$ under the key $\mathbf{s}_1, \mathbf{s}_2$ with noise $\mathbf{e}_1, \mathbf{e}_2$, the tuple $(\mathbf{A}, \mathbf{c}_1 + \mathbf{c}_2)$ is an encryption of $\mathbf{x}_1 + \mathbf{x}_2$ under the key $\mathbf{s}_1 + \mathbf{s}_2$. The ciphertext $(\mathbf{A}, \mathbf{c}_1 + \mathbf{c}_2)$ can be properly decrypted if $\mathbf{e}_1 + \mathbf{e}_2$ is small. Note that computing $\mathbf{c}_1 + \mathbf{c}_2$ is very efficient—it is simply vector addition.

For ease of presentation later, we define a tuple of algorithms $(\mathsf{Enc}, \mathsf{Dec})$ about public parameters $(p, q, \lambda, m, \mathbf{A} \in \mathbb{Z}_q^{m \times \lambda})$ as follows:

- $\mathsf{Enc}(\mathbf{s}, \mathbf{x}) \to \mathbf{y}$: on input a secret key $\mathbf{s} \in \mathbb{Z}_q^\lambda$ and a message $\mathbf{x} \in \mathbb{Z}_q^m$, output $\mathbf{y} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{x}$, where $\Delta = \lfloor q/p \rfloor$.
- $\mathsf{Dec}(\mathbf{s}, \mathbf{y}) \to \mathbf{x}'$: on input a secret key $\mathbf{s} \in \mathbb{Z}_q^\lambda$ and a ciphertext $\mathbf{y} \in \mathbb{Z}_q^m$, output $\mathbf{x}' := \lfloor \mathbf{y} - \mathbf{A}\mathbf{s} \rceil_\Delta$.

Packed secret sharing. In standard Shamir secret sharing [61], one picks a secret s and generate a polynomial $f(x) = a_0 + a_1 x + \ldots + a_t x^d$ where $a_0 = s$ and $a_1, \ldots, a_d$ are random. Assuming there are n parties, the share for party $i \in [n]$ is $f(i)$, and any subset of at least $d + 1$ parties can reconstruct s and any subset of d shares are independently random.

In packed secret sharing [33], one can hide multiple secrets using a single polynomial. Specifically, let $\mathbb{F}$ be a field of size at least $2n$ and k be the number of secrets packed in one sharing. Packed Shamir secret sharing of $(s_1, \ldots, s_k) \in \mathbb{F}^k$ first chooses a random polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most $d + k - 1$ subject to $f(0) = s_1, \ldots, f(-k+1) = s_k$, and then sets the share v_i for party i to be $v_i = f(i)$ for all $i \in [n]$. Reconstruction of a degree-$(d + k - 1)$ sharing requires at least $d + k$ shares from $v_1, \ldots, v_n$. Note that now the corruption threshold is d, i.e., any d shares are independently random but any $d + 1$ shares are not.

Shamir share testing. Looking ahead, we will also use a probabilistic test for Shamir's secret shares, called SCRAPE test [21]. To check if $(s_1, \ldots, s_n) \in \mathbb{F}^n$ is a Shamir sharing over $\mathbb{F}$ of degree d (namely there exists a polynomial p of degree $\leq d$ such that $p(i) = s_i$ for $i = 1, \ldots, n$), one can sample $w_1, \ldots, w_n$ uniformly from the dual code to the Reed-Solomon code formed by the evaluations of polynomials of degree $\leq d$, and check if $w_1 s_1 + \ldots + w_n s_n = 0$ in $\mathbb{F}$. If the test passes, then $s_1, \ldots, s_n$ are Shamir Shares, except with probability $1/|\mathbb{F}|$.

Specifically, for a finite field $\mathbb{F}$ and given parameters d, n such that $0 \leq d \leq n - 2$, and inputs $s_1, \ldots, s_n \in \mathbb{F}$. Let $v_i := \prod_{j \in [n] \setminus \{i\}} (i - j)^{-1}$ and $m^*(X) := \sum_{i=0}^{n-d-2} m_i \cdot X^i \leftarrow_\$ \mathbb{F}[X]_{\leq n-d-2}$ (i.e., a random polynomial over the field of degree at most $n - d - 2$). Now, let $\mathbf{w} := (v_1 \cdot m^*(1), \ldots, v_n \cdot m^*(n))$ and $\mathbf{s} := (s_1, \ldots, s_n)$. Then,

- If there exists $p \in \mathbb{F}[X]_{\leq d}$ such that $s_i = p(i)$ for all $i \in [n]$, then $\langle \mathbf{w}, \mathbf{s} \rangle = 0$.

- Otherwise, $\Pr[\langle \mathbf{w}, \mathbf{s} \rangle = 0] = 1/|\mathbb{F}|$.

Pedersen and vector commitment. Let $\mathbb{G}$ be a group of order q, and G, H be two generators in $\mathbb{G}$. A Pedersen commitment to a value $v \in \mathbb{Z}_q$ is computed as $\mathsf{com}_G(v) := G^v H^r$, where r is the commitment randomness, uniformly chosen from $\mathbb{Z}_q$. We use $\mathsf{com}_G(\cdot)$ notation because later in our protocol, we will compute commitments with different generators.

We can also commit to a vector $\mathbf{v} = (v_1, \ldots, v_L) \in Z_q^L$ as follows: let $\mathbf{G} = (G_1, \ldots, G_L)$ be a list of L random generators in $\mathbb{G}$, define $\mathsf{com}_{\mathbf{G}}(\mathbf{v}) := G_1^{v_1} \cdots G_L^{v_L} \cdot H^r$, where r is randomly chosen from $\mathbb{Z}_q$.

Inner-product proof. The inner-product proof allows a prover to convince a verifier that, given vector commitments to two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_q^L$, and a public value c, the prover knows the opening of the commitments such that $\langle \mathbf{a}, \mathbf{b} \rangle = c$. Bulletproof [17] and its later variants [36] give inner-product proof in one round (using Fiat-Shamir), with proof size $O(\log L)$ and prover/verifier cost $O(L)$.

For ease of presentation later, we introduce the following notations for proof. A proof system Π consists of a tuple of algorithms $(\mathcal{P}, \mathcal{V})$ run between a prover and verifier. An argument to prove can be described with public inputs/outputs io, a statement to be proved st, and a witness wt. Given a proof system Π, the prover can generate a proof $\pi \leftarrow \Pi.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$ and the verifier checks the proof by $b \leftarrow \Pi.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi)$ where $b \in \{0, 1\}$ indicates rejecting or accepting π. For example, for proving inner product of $\mathbf{a}$ and $\mathbf{b}$, we set the constraint system to be

$$\{\mathsf{io} : (\mathsf{com}(\mathbf{a}), \mathsf{com}(\mathbf{b}), c),\ \mathsf{st} : \langle \mathbf{a}, \mathbf{b} \rangle = c,\ \mathsf{wt} : (\mathbf{a}, \mathbf{b})\}.$$

Denote the inner product proof system (e.g., Bulletproof [17]) as Π_{ip}, the prover runs $\pi \leftarrow \Pi_{\text{ip}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$ and the verifier runs $b \leftarrow \Pi_{\text{ip}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$. The algorithms $\Pi_{\text{ip}}.\mathcal{P}$ and $\Pi_{\text{ip}}.\mathcal{V}$ both has complexity linear to the length of $\mathbf{a}$ (or $\mathbf{b}$) and π has logarithmic length of $\mathbf{a}$ (or $\mathbf{b}$). Later we will also use an optimized inner-product proof when $\mathbf{a}$ is public, called *linear-relation proof*; in this case, the constraint system will be

$$\{\mathsf{io} : (\mathsf{com}(\mathbf{b}), c),\ \mathsf{st} : \langle \mathbf{a}, \mathbf{b} \rangle = c,\ \mathsf{wt} : \mathbf{b}\}.$$

B Related Work

Single-server setting. Bonawitz et al. [13] gives the first dropout-resilient secure aggregation protocol for federated learning. Subsequently, a line of work [8,64,65,51,37,48] focuses on improving the efficiency of this protocol. Recently, there has been growing interest in ensuring input validity inside secure aggregation, and we briefly review the techniques used in prior work.

Eiffel [24] uses SNIP [25] to prove arbitrary predicate on inputs but with high communication. Specifically, each client secret-shares its input vector to other clients who act as the multiple verifiers in SNIP. RoFL [50] adopts the protocol by Bonawitz et al. [13], and uses range proof (and hence does not work for arbitrary predicates) to bound the norms of input vectors. RoFL's communication cost is significantly less than Eiffel but is still expensive as the client in RoFL proves the range for *each* entry of the input vector which requires sending ℓ Pedersen commitments to the server for a vector of length ℓ. Readers can refer to a comprehensive comparison of communication costs in Bell et al. [9, Table 1].

The most relevant work to ours is ACORN family [9], where they present two protocols, ACORN-detect which have the same property as RoFL, and ACORN-robust which additionally has robustness. ACORN-detect reduces the expensive m commitments (for range proof on a length-m vector) to a single commitment using the technique of approximate proof [36]. They extend ACORN-detect to ACORN-robust but with the price of a significant increase in rounds: after aggregating the inputs, they run an $O(\log n)$-round protocol between the server and clients to identify the cheaters and remove their inputs from the sum.

Armadillo has four rounds only, but the tradeoff is that a small set of clients will need to do $O(n)$ work (though concretely fast); meanwhile, ACORN-robust has $O(\text{polylog } n)$ work per client. This tradeoff is meaningful if one considers concrete parameters (Section 1.1) since each client anyway needs to do work proportional to ℓ (input vector length) if that is already larger than n.

For completeness, we also briefly survey related literature in the multi-server setting.

Two (or more) servers. There are also works that split trust across multiple servers, like the two-server solutions Elsa [58] and SuperFL [67] or the generic multi-server solution Flag [7]. These works have the clients secret share their input vectors to two or more servers, and the servers communicate with each other to validate the inputs. These solutions are more efficient in terms of run time compared to the single-server ones. However, ensuring non-collusion among communicating servers is the major source of criticism against these solutions, aside from the obvious overhead of deploying multiple servers.

To be precise, in the multi-server setting, the servers are powerful machines and thus can execute heavy computation (e.g., $O(n\ell)$ for the secret-sharing-based solutions [58,7] where n is the number of clients and ℓ is the vector length); in the single-server setting, the heavy computation is pushed to the only server and all the clients are restricted in both computation and communication. In other words, any protocol incurring $O(n\ell)$ cost at any client is not an effective single-server protocol. In Armadillo each client has cost $O(n+\ell)$; in practice, ℓ is much larger than n, so the cost will be dominated by ℓ which is the input size.

Prior work [4,51] gives detailed discussion and formalization for this model, where the trust is split across a small set of clients with restricted power, but they can still help the server aggregation with reasonable cost. Crucially, a helper is also a client; and in our protocol, they just do slightly more work than the regular clients.

C Deferred Material for Zero-knowledge Proof

Proof of Shamir sharing. To be precise why we need such proof: suppose each client should share its key (i.e., $\mathbf{k}_i$) using a degree-d polynomial, but a malicious client shares its key using a polynomial of degree higher than d. Later when the server collects the shares from the helpers, the server cannot interpolate the shares to a degree-d polynomial and hence the inner aggregation fails.

A natural approach is to use a verifiable secret sharing (VSS) (eg.[31]), where each client acts as the dealer who shares $\mathbf{k}_i$ to the helpers, and the helpers themselves run the VSS to identify malicious dealers (and exclude their shares from inner aggregation). This either requires interaction between the helpers (e.g., if using BGW [11]), or heavy computation cost at the helpers (e.g., if using Feldman protocol [30]).

We instead use the SCRAPE test (Section 2). Suppose client i has a sharing $\mathbf{s}_i = (s_i^{(1)}, \ldots, s_i^{(C)})$, which the client claims is Shamir sharing of a prescribed degree d over $\mathbb{F}$. Now, the client commits to $\mathbf{s}$ using vector commitment and then invokes a linear-relation proof that

$$\langle \mathbf{s}_i, \mathbf{w} \rangle = 0 \text{ in } \mathbb{F},$$

where $\mathbf{w} := (w_1, \ldots, w_n)$ is sampled uniformly random from some code space (details in Section 2). In our setting, we cannot let the client choose $\mathbf{w}$ (since they can be malicious), so we apply the Fiat-Shamir transform and have the client derive $\mathbf{w}$ by hashing the commitment to $\mathbf{s}_i$.

As long as we assume the secrets are correctly shared, and assuming $\delta_{\mathcal{C}} + \eta_{\mathcal{C}} < 1/3$, the server can always reconstruct $\mathbf{k}$ successfully (with Berlekamp-Welch algorithm).

Binding $\mathbf{k}_i$ in inner and outer aggregation. Ensuring that $\mathbf{s}_i$ is a Shamir sharing is not sufficient, we also need to ensure that $\mathbf{s}$ is a sharing of $\mathbf{k}_i$ that was committed to. For this, we can also use a linear proof, constructed as follows.

For each client i, let $\mathbf{s}_i = (s_i^{(1)}, \ldots, s_i^{(C)})$ be the C Shamir shares of $\mathbf{k}_i$, and the shared polynomial has coefficients $\mathbf{a} = (a_0, \ldots, a_d)$. Let both $\mathbf{s}_i$ and $\mathbf{a}$ be column vectors. We have $\lambda < d < C$, and there exists a public matrix $\mathbf{M} \in \mathbb{Z}_q^{\lambda \times C}$ such that

$$\mathbf{M} \cdot \mathbf{s}_i = \mathbf{k}_i.$$

This can be proved using a linear-relation inner-product proof, shown in Figure 6 and detailed at the end of this section.

A final complication is that each helper j needs to check if the received share $s_i^{(j)}$ is what the client committed to—this is because the communication between clients is using end-to-end symmetric encryption (see Section 1.1), and a malicious client could send to the helper j a share $s_i^{(j)}$ that is not

Constraint system $\mathbb{CS}$:

$$\begin{aligned}\{&\mathsf{io} : (\mathsf{com}(\mathbf{v}_1), \mathsf{com}(\mathbf{v}_2), \mathbf{M}),\\ &\mathsf{st} : \mathbf{v}_2 = \mathbf{M} \cdot \mathbf{v}_1,\\ &\mathbf{wt} : (\mathbf{v}_1, \mathbf{v}_2)\}, \text{ where } \mathbf{v}_1, \mathbf{v}_2 \in \mathbb{Z}_q^L, \mathbf{M} \in \mathbb{Z}_q^{W\times L}\end{aligned}$$

Let $r \leftarrow \mathcal{H}(\mathsf{com}(\mathbf{v}_1), \mathsf{com}(\mathbf{v}_2))$, where $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q$.
Let $\mathbf{r} := (r^0, r^1, \ldots, r^{W-1})$.
Set constraint system $\mathbb{CS}'$:

$$\begin{aligned}\{&\mathsf{io}' : (\mathsf{com}(\mathbf{v}_1) \cdot \mathsf{com}(\mathbf{v}_2),\ 0)\\ &\mathsf{st}' : \langle \mathbf{M}^\top \mathbf{r} - \mathbf{r},\ \mathbf{v}_1 | \mathbf{v}_2 \rangle = 0,\\ &\mathsf{wt}' : \mathbf{v}_1 | \mathbf{v}_2\}.\end{aligned}$$

$\Pi_{\text{linear}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$: Output $\Pi_{\text{ip}}.\mathcal{P}(\mathsf{io}', \mathsf{st}', \mathsf{wt}')$.
$\Pi_{\text{linear}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi)$: Output $\Pi_{\text{ip}}.\mathcal{V}(\mathsf{io}', \mathsf{st}', \pi)$.

Figure 6: Protocol Π_{linear} proves matrix-vector multiplication, built on inner-product proof protocol Π_{ip}.

consistent with the commitment. To prevent this, we let client i send to each helper j the following messages:

1. The commitments to the shares, $\mathsf{com}_{G_j}(s_i^{(j)}) := G_j^{s_i^{(j)}} \cdot H^{r_j}$, where H, G_j are group generators. Note that for different helper j, the generator G_j used for commitment is different. The commitments are sent in the clear.
2. The openings to the commitment, namely the commitment randomness r_j and the actual share $s_i^{(j)}$, symmetrically encrypted.

Note that the vector commitment to $\mathbf{s}_i$ will be directly derived from the individual commitments to the shares, i.e., given $\mathsf{com}_{G_1}(s_i^{(1)}), \ldots, \mathsf{com}_{G_C}(s_i^{(C)})$ where the underlying randomness are $r_1, \ldots, r_C$ respectively, the vector commitment to $\mathbf{s}_i$ is computed as $\prod_{j=1}^{C} \mathsf{com}(s_i^{(j)})$ with randomness $r = \sum_{j=1}^{C} r_j$.

Proof of linear relation. Now we specify the details for the proof of linear relation. Given commitments to vectors $\mathbf{v}_1, \mathbf{v}_2$ and a public matrix $\mathbf{M}$, we show how to prove $\mathbf{v}_2 = \mathbf{M} \cdot \mathbf{v}_1$ using a single inner-product proof by Schwartz-Zippel Lemma. We first rewrite the statement as $\mathbf{M}\mathbf{v}_1 - \mathbf{v}_2 = \mathbf{0}$, where $\mathbf{0}$ is a zero vector. The idea is to view the vector as coefficients of a polynomial and check if the evaluation of a random point on the polynomial gives 0. Specifically, suppose $\mathbf{M}$ has W rows, and let r be a random value in $\mathbb{Z}_q$ and let $\mathbf{r} = (r^0, r^1, \ldots, r^{W-1})$. We transform the matrix-vector multiplication into linear combinations of inner products:

$$\langle \mathbf{M}^\top \mathbf{r}, \mathbf{v}_1 \rangle + \langle -\mathbf{r}, \mathbf{v}_2 \rangle = \langle \mathbf{M}^\top \mathbf{r} | (-\mathbf{r}), \mathbf{v}_1 | \mathbf{v}_2 \rangle = 0.$$

If $\mathbf{r}$ is chosen after $\mathbf{v}_1$ and $\mathbf{v}_2$ are committed, then we can be sure (except with probability W/q) that $\mathbf{M}\mathbf{v}_1 = \mathbf{v}_2$ holds as long as the above equation holds. Also, note that the verifier can compute $\mathsf{com}(\mathbf{v}_1 | \mathbf{v}_2)$ as $\mathsf{com}(\mathbf{v}_1) \cdot \mathsf{com}(\mathbf{v}_2)$. Figure 6 formally shows the protocol.

C.1 Proof of Encryption and Input Validity

In this section, we describe how to 1) prove the encryption is computed correctly and 2) the input vector has a bounded L_2, L_∞ norm.

The first part is to prove $\mathbf{y}_i = \mathsf{Enc}(\mathbf{k}_i, \mathbf{x}_i)$ is correctly computed, i.e., we want to prove that, given commitment to $\mathbf{x}_i, \mathbf{k}_i, \mathbf{e}_i$, and a public $\mathbf{y}_i$, there is $\mathbf{y}_i = \mathbf{A}\mathbf{k}_i + \mathbf{e}_i + \lfloor q/p \rfloor \cdot \mathbf{x}_i$ and $\mathbf{e}_i$ has small L_∞ norm. We next break this down into several proofs, some of which will also be useful for proving input validity.

Proof of L_∞ norm. We first explain the reason why trivially applying range proof (such as Bulletproof [17]) to each vector component will not work well in our setting. Let us recall how Bulletproof proves range for a single value: say we want to prove $v \in [0, 2^B - 1]$, then the prover decomposes v into B binary values, denoted as $\mathbf{a} \in \mathbb{Z}_2^B$; and let $\mathbf{b} = (2^0, 2^1, \dots, 2^{B-1})$ be a public vector. Then the prover proves that $\langle \mathbf{a}, \mathbf{b} \rangle = v$, and proves that every entry of $\mathbf{a}$ is in $\{0, 1\}$. This approach has the cost growing with range size for each entry—if the range size is large, say 2^{16}, then the prover needs to decompose each value into 16 binary values. This is efficient when the prover only proves range on a small number of values, however, in the federated learning setting, the client needs to prove ℓ values (ℓ is the vector length), meaning that the client needs to compute $B\ell$ commitments. Since ℓ is large (see concrete examples in Section 1.1), even B is small like 16, computing 16ℓ Pedersen commitment is already a high cost for the client.

We use a technique by Bell et al. [9] which builds a range proof on vectors with a cost of only $O(\ell)$.

Given $\mathbf{a}$ of length m, we want to prove that $\|\mathbf{a}\|_\infty < B$. It is then reduced to proving the following statements:

- The prover defines $\mathbf{a}' = 2\mathbf{a} - (B-1)\mathbf{1}$ and finds $\mathbf{u}, \mathbf{v}, \mathbf{w}$ and proves that $\mathbf{a}' \circ \mathbf{a}' + \mathbf{u} \circ \mathbf{u} + \mathbf{v} \circ \mathbf{v} + \mathbf{w} \circ \mathbf{w} = -(B^2 - 2B + 2)\mathbf{1}$,[5]
- The prover proves $\|\mathbf{a}' | \mathbf{u} | \mathbf{v} | \mathbf{w}\|_\infty < \sqrt{q}/4$.

The first part can be reduced into an inner product using Schwartz-Zippel Lemma. Let r be a random value chosen after the witness $\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{a}'$ are committed, and let $\mathbf{r} := (r^0, r^1, \dots, r^{m-1})$. If the prover can prove the following relation,

$$\langle \mathbf{a}', \mathbf{a}' \circ \mathbf{r} \rangle + \langle \mathbf{u}, \mathbf{u} \circ \mathbf{r} \rangle + \langle \mathbf{v}, \mathbf{v} \circ \mathbf{r} \rangle + \langle \mathbf{w}, \mathbf{w} \circ \mathbf{r} \rangle = \langle \mathbf{c}, \mathbf{r} \rangle$$

where $\mathbf{c} = -(B^2 - 2B + 2)\mathbf{1}$, then the original relation holds except probability m/q. The above proof can be further reduced to a single inner-product proof $\langle \mathbf{z}_1, \mathbf{z}_2 \rangle = c$ for some $\mathbf{z}_1, \mathbf{z}_2$ of length $4m$ and a public value c [9,36].

The second part requires again a proof of L_∞, but the essence is that it is a *loose* range proof, where the actual entries in $\mathbf{a}'$ (similarly $\mathbf{u}, \mathbf{v}, \mathbf{w}$) are much smaller than the bound $\sqrt{q}/4$. This is exactly *approximate proof*, introduced by Gentry et al. [36]: given a vector $\mathbf{b}$ of length m' where $\|\mathbf{b}\|_\infty < B$, we aim to prove $\|\mathbf{b}\|_\infty < B'$ where $B \lll B'$, which is much easier than proving $\|\mathbf{b}\|_\infty < B$. They give a protocol that proves $\|\mathbf{b}\|_\infty < B'$ using only a single inner-product proof of length $m' + \sigma$ where σ is a security parameter (see details in Appendix C.2). In our case, we just set $\mathbf{b} = \mathbf{a}' | \mathbf{u} | \mathbf{v} | \mathbf{w}$ and correspondingly $m' = 4m$.

In sum, proving L_∞ norm of a length-m vector requires a length-$4m$ inner-product proof (the first part), and a length-$(4m + \sigma)$ inner-product proof (the second part) where σ is a security parameter typically taken as 256.

Proving L_2 norm. Suppose the prover has a length-m vector $\mathbf{a}$ and wishes to prove $\|\mathbf{a}\|_2 < B$. The prover finds four non-negative integers $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ such that $(\alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2) + \|\mathbf{a}\|_2 = B^2$. Let $\mathbf{u} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ and $\mathbf{v} = (\mathbf{a} | \mathbf{u})$. The prover does an inner product proof that $\|\mathbf{v}\|_2 = B^2$. Also, the prover does an approximate proof that $\|\mathbf{a}\|_\infty < \sqrt{q/(m+4)}$.

We can reduce the cost of the quadratic proof $\|\mathbf{v}\|_2 = B^2$ using a matrix projection technique from Gentry et al. [36]: this reduces proving L_2 norm on a long vector into proving L_2 norm on a size-256 vector. Given a vector $\mathbf{a}$ of length m, sample a matrix $\mathbf{R} \leftarrow \mathcal{D}^{256 \times m}$ from a special distribution[6] $\mathcal{D}$, if $\mathbf{b} := \mathbf{R}\mathbf{a}$ has small L_2 norm, then with high probability $\mathbf{a}$ also has small L_2 norm. Therefore, we just need to invoke L_2 proof on $\mathbf{b}$.

The above projection technique is correct when we work over integers, but if we work over $\mathbb{Z}_q$, $\mathbf{a}$ may have a large L_2 norm but $\mathbf{b}$ has a small L_2 norm. But this event can only occur when the entry of $\mathbf{a}$ is large enough so that when multiplied with $\mathbf{R}$, the values get wrapped around in $\mathbb{Z}_q$. Since $\mathbf{R}$

[5] This is also known as Lagrange's four-square theorem. Rabin and Shallit proposed randomized algorithms for computing a single representation for a given integer a as $a = \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2$ in $O(\log^2 a)$ time [57].

[6] The distribution $\mathcal{D}$ is: $\mathcal{D}(0) = 1/2$ and $\mathcal{D}(\pm 1) = 1/4$. Since a sample from $\mathcal{D}$ is binary, transmitting matrices $\mathbf{R}$ and $\mathbf{R}'$ incur very small communication costs. The row dimension 256 is chosen by Johnson-Lindenstrauss lemma [40] to ensure checking on the projected (short) vector is sufficient except with negligible probability.

consists of entries only from $\{-1, 0, 1\}$, we just still need the above approximate proof for $\mathbf{a}$ to show that wrapping around does not happen.

Putting things together. We now describe the proof of Regev's encryption Π_{enc} ([36, Lemma 3.7]). For now, we set the ciphertext modulus q in LWE equal to the group size in the commit-and-proof system for ease of presentation. In Section E, we will discuss how to handle different ciphertext modulus and proof system modulus.

Recall that the constraints that client i wishes to prove is

$$\begin{aligned}\mathbb{CS} : \{&\mathsf{io} : (\mathsf{com}(\mathbf{k}_i), \mathsf{com}(\mathbf{x}_i), \mathsf{com}(\mathbf{e}_i)),\\ &\mathsf{st} : \mathbf{y}_i = \mathbf{A} \cdot \mathbf{k}_i + \mathbf{e}_i + \lfloor q/p \rfloor \mathbf{x}_i,\\ &\qquad \|\mathbf{e}_i\|_2 < B_{\mathbf{e}}, \|\mathbf{x}_i\|_2 < B_{\mathbf{x}}(L_2), \|\mathbf{x}_i\|_\infty < B_{\mathbf{x}}(L_\infty),\\ &\mathsf{wt} : (\mathbf{k}_i, \mathbf{x}_i, \mathbf{e}_i)\}\end{aligned}$$

Since already described how to prove the L_∞ bound of $\|\mathbf{x}_i\|_\infty$, so we omit it here. For simplicity, below we omit the subscript i. Protocol Π_{enc} works as follows:

1. The prover sets $\mathbf{y} = \mathbf{A}\mathbf{k} + \mathbf{e} + \lfloor q/p \rfloor \mathbf{x} \mod q$, sends to the verifier $\mathbf{y}$ and the commitment to $\mathbf{k}, \mathbf{x}, \mathbf{e}$. Recall that $\mathbf{A} \in \mathbb{Z}_q^{m \times \lambda}, \mathbf{e} \in \chi^m$.
2. The verifier chooses projection matrices $\mathbf{R} \leftarrow \mathcal{D}^{256 \times \lambda}$ and $\mathbf{R}' \leftarrow \mathcal{D}^{256 \times m}$, and sends them to the prover.
3. The prover computes $\mathbf{u} := \mathbf{R}' \cdot \mathbf{e}$, and $\mathbf{v} = \mathbf{R}' \cdot \mathbf{x}$. The prover aborts if $\|\mathbf{u}\|_2 > B_{\mathbf{u}}$ or $\|\mathbf{v}\|_2 > B_{\mathbf{v}}$, otherwise it sends to the verifier the commitment to $\mathbf{u}, \mathbf{v}$. The bound $B_{\mathbf{u}}, B_{\mathbf{v}}$ are determined by the LWE parameters and the bound $B_{\mathbf{x}}, B_{\mathbf{e}}$. Note that vectors $\mathbf{u}, \mathbf{v}$ are only of length 256.
4. The prover and the verifier run the following sub-protocols:
   (a) Proof of L_2 norm that
   $$\|\mathbf{u}\|_2 < B_{\mathbf{u}},\ \|\mathbf{v}\|_2 < B_{\mathbf{v}}$$
   (b) Proof of linear relation that:
   $$\mathbf{R}' \cdot \mathbf{e} = \mathbf{u}, \quad \mathbf{R}' \cdot \mathbf{x} = \mathbf{v} \mod q.$$
   (c) Proof of linear relation that:
   $$\mathbf{R}' \cdot \mathbf{y} = (\mathbf{R}'\mathbf{A}) \cdot \mathbf{k} + \mathbf{u} + \lfloor q/p \rfloor \mathbf{v} \mod q.$$
   (d) Approximate proof that:
   $$\|\mathbf{x}\|_\infty, \|\mathbf{e}\|_\infty < \sqrt{q/(m+4)}$$
5. The verifier accepts if all the above proofs pass.

For step 4(a), we can directly prove L_2 norm as we described before for length-256 vectors $\mathbf{u}, \mathbf{v}$. For step 4(b), we can directly invoke the linear proof (Figure 6). Step 4(c) can be proved by showing

$$\langle (\mathbf{R}'\mathbf{A})^\top \mathbf{r}, \mathbf{k} \rangle + \langle \mathbf{r}, \mathbf{u} \rangle + \langle \lfloor q/p \rfloor \mathbf{r}, \mathbf{v} \rangle = \langle \mathbf{R}'\mathbf{b}, \mathbf{r} \rangle,$$

where $\mathbf{r}$ is powers of a random value r as before; and the sum on the LHS is in fact

$$\langle (\mathbf{R}'\mathbf{A})^\top \mathbf{r} | \mathbf{r} | \lfloor q/p \rfloor \mathbf{r},\ \mathbf{k} | \mathbf{u} | \mathbf{v} \rangle,$$

and note that the verifier can compute the commitment to $\mathbf{k}|\mathbf{u}|\mathbf{v}$ as $\mathsf{com}(\mathbf{k}) \cdot \mathsf{com}(\mathbf{u}) \cdot \mathsf{com}(\mathbf{v})$. For step 4(d), we use the approximate proof described before (see details in Appendix C.2).

C.2 Approximate proof

For completeness, we describe the protocol in Gentry, Halevi, and Lyubashevsky [36] below. Let the security parameter be σ. The prover has a vector $\mathbf{a}$ of length m where $\|\mathbf{a}\|_\infty < B$. Let B' be the bound that the prover can prove with the following protocol. For security, the gap $\gamma := B'/B$ should be larger than $19.5\sigma\sqrt{m}$.

1. The prover first sends $\mathsf{com}(\mathbf{a})$ to the verifier.
2. The prover chooses a uniform length-σ vector $\mathbf{y} \xleftarrow{\$} [\pm\lceil b/2(1+1/\sigma)\rceil]^{\sigma}$, and sends $\mathsf{com}(\mathbf{y})$ to the verifier.
3. The verifier chooses $\mathbf{R} \leftarrow \mathcal{D}^{\sigma \times m}$ and sends it to the prover.
4. The prover computes $\mathbf{u} := \mathbf{R} \cdot \mathbf{a}$ and $\mathbf{z} = \mathbf{u} + \mathbf{y}$. It restarts the protocol from Step 2 if either $\|\mathbf{u}\|_\infty > b/2\lambda$ or $\|\mathbf{z}\| > b/2$.
5. The prover sends $\mathbf{z}$ to the verifier.
6. The verifier chooses a random r and sends r to the prover.
7. The prover and the verifier run an inner-product proof that

$$\langle \mathbf{R}^\top \mathbf{r}, \mathbf{a} \rangle + \langle \mathbf{r}, \mathbf{y} \rangle = \langle \mathbf{R}^\top \mathbf{r} | \mathbf{r}, \mathbf{a} | \mathbf{y} \rangle = \langle \mathbf{z}, \mathbf{r} \rangle,$$

where $\mathbf{r} = (r^0, r^1, \ldots, r^{\sigma-1})$.

Note that $\langle \mathbf{z}, \mathbf{r} \rangle$ is a public value. The last step is essentially a length-$(m+\sigma)$ inner product proof.

C.3 Proof of Theorem 1

Below we analyze the number of inner-product proof (IP) invocations required for a client. Proving Shamir shares requires an IP of length C. Proving binding relation requires an IP of length $C + \lambda$. Proving L_∞ norm requires two IPs of length $4m$. Proving the encryption requires two IPs of length m for step 4(b), one IP of length λ for step 4(c), and two IPs of length m for step 4(d).

C.4 Other details

Selecting helpers. We can select helpers in two different ways, based on different assumptions. If assuming a random beacon, one can follow the approach in Flamingo [51], where the helpers are determined by the randomness generated from a beacon. Assuming the corrupted rate of the total population N, and fix a target $\eta_{\mathcal{C}}$, then one can derive C with N and $\eta_{\mathcal{C}}$ (Appendix C, [51]).

If assuming a public bulletin board, one can use Feige's election protocol [29] to select the set of helpers: we initialize a certain number of bins on the bulletin board, and each client chooses to jump in a bin independently random (malicious clients may not do it randomly). Then we take the bin of the smallest number of clients as the helper set. The advantage of the Feige protocol is that when the total population has a corruption rate η, the sampled set also has a corruption rate at most η.

Selecting participants per iteration. Several works discuss attacks orthogonal to the cryptographic design, and we discuss how to mitigate them in our system. So et al. [63] demonstrate that the server can infer some clients' data if it observes the sums from many rounds of aggregation, even if each round the participants are selected at random. They proposed a selection strategy called batch partitioning, with the idea of restricting the clients into certain batches that either participate together or do not participate at all.

When the server is semi-honest, we can let the server follow this selection strategy. When the server is malicious, we ask the helpers to cross-check if the participating clients conform with the selection algorithm.

Pasquini et al. [55] show another attack where a malicious server can elude secure aggregation by sending clients inconsistent models. Prior work on secure aggregation [51] proposes mitigation that prevents the server from learning anything if it sends inconsistent models; their key idea is to bind the hash of the model to the pairwise masks which are canceled out if all the clients have the same hash. Here we use a different approach: let the client hash the received model and send the hash to the helpers, then the helpers do a majority vote on the hashes and exclude the shares from the clients whose hashes do not equal the majority vote.

Privacy against malicious server. So far we only consider a semi-honest server. A malicious server can ask the helpers for any set of which it wishes to know the sum: say the server wants to target for client $i \in S$, it asks $d+1$ helpers for aggregating shares for the set S and asks another $d+1$ helpers for aggregating the shares for the set $S \backslash \{i\}$. What we can guarantee is that the server learns the sum from a sufficiently large set of size $(1-\delta-\eta)n$ by having the helpers cross-check the online set (Round 2 in Figure 5): if the majority of the helpers agrees on the online set, they will continue the protocol; otherwise they abort. We formally state the malicious security in Section D.

We give the full details for a single aggregation in Figure 5. There are three proof sub-protocols we use: Π_{ip} (Section 2), Π_{linear}(Figure 6), Π_{enc} (Section C.1). The protocol works in three rounds: in the first round, the server collects the encrypted inputs from all the clients, and in the second and third round, the server talks to only the helpers who compute an aggregate key for the server to decrypt the aggregate ciphertext.

D Security analysis

In this section, we discuss how to select proper parameters for our protocol, and formally state the properties of Armadillo.

Parameters. The system Armadillo has a set of parameters listed below. First, n is the number of clients per round. (λ, m, p, q) are LWE parameters (for the outer aggregation), (C, d, a) are secret-sharing parameters (for the inner aggregation), where C is the number of helpers selected, d is the degree of the secret-sharing polynomial and a is the number of packed secrets. In our protocol, $a = \lambda$. Finally, $\mathbb{G}$ is the group (of order q) for commit-and-proof system, and $B_{\mathbf{x}}(L_\infty), B_{\mathbf{x}}(L_2), B_{\mathbf{e}}$ are bounds on norms.

The parameters $n, B_{\mathbf{x}}(L_\infty), B_{\mathbf{x}}(L_2), B_{\mathbf{e}}$ are chosen depending on the machine learning setting, which is orthogonal to security analysis. For (λ, m, p, q), we can choose any secure instance of LWE and we can refer to recent security analysis [3,26,22].

For C, we need to choose according to the LWE parameters together with the dropout rate $\delta_{\mathcal{C}}$ and malicious rate $\eta_{\mathcal{C}}$. Recall that in our protocol (Section 3), each client secret-shares a short vector of length λ (with packed secret sharing) using a polynomial of degree d. We must have

$$\begin{aligned} &d - \lambda > C \cdot \eta_{\mathcal{C}} && \text{by security of packed secret sharing,} \\ &d < C(1 - \delta_{\mathcal{C}}) && \text{in order to reconstruct the secret.} \end{aligned}$$

Combining these two equations, we get $C > \lambda/(1 - \delta_{\mathcal{C}} - \eta_{\mathcal{C}})$. Also note that we have $\delta_{\mathcal{C}} + \eta_{\mathcal{C}} < 1/3$ as the assumption required for robustness (Section 3.2), we have $\lambda/(1 - \delta_{\mathcal{C}} - \eta_{\mathcal{C}}) < 3\lambda/2$; therefore, setting $C \geq 3\lambda/2$ is sufficient, and accordingly, we set d to be

$$(1 + \frac{3}{2}\eta_{\mathcal{C}})\lambda < d < \frac{3}{2}\lambda(1 - \delta_{\mathcal{C}}).$$

Now we formally state the properties of Armadillo. Let Φ denote the protocol in Figure 5 (private sum for a single training iteration). In particular, Φ is a protocol running between n clients and the server, where each client i has inputs $\mathbf{x}_i \in \mathbb{Z}_q^m$ and the server has no input. We describe the ideal functionality of Φ in Figure 7.

Theorem 2 (Dropout resilience of Φ)**.** Let $(\delta, \eta, \delta_{\mathcal{C}}, \eta_{\mathcal{C}})$ be threat model parameters defined in Section 1.1. Let (C, d) be parameters for the secret-sharing protocol (the inner aggregation). If $d < C(1 - \delta_{\mathcal{C}})$, then protocol Φ (Figure 5) satisfies dropout resilience: on input $\mathbf{x}_i$ for client $i \in [n]$, when the n clients and the server follow protocol Φ, given a dropout set $\mathcal{O} \subset [n]$ (where $|\mathcal{O}| < \delta n$) at any point of the execution, protocol Φ will terminate and output $\sum_{i \in [n] \setminus \mathcal{O}} \mathbf{x}_i$.

Theorem 3 (Privacy and robustness of Φ)**.** Let $(\delta, \eta, \delta_{\mathcal{C}}, \eta_{\mathcal{C}})$ be threat model parameters defined in Section 1.1. Let (λ, m, p, q) be LWE parameters, and let (C, d, λ) be the parameters of packed secret sharing. If (λ, m, p, q) is a secure instance of LWE, and $\delta_{\mathcal{C}} + \eta_{\mathcal{C}} < 1/3$, $C \geq 3\lambda/2$, $(1 + \frac{3}{2}\eta_{\mathcal{C}})\lambda < d < \frac{3}{2}\lambda(1 - \delta_{\mathcal{C}})$, then under the communication model defined in Section 1.1, assuming PKI, and a random beacon (or a public bulletin board), protocol Φ (Figure 5) securely realizes ideal functionality $\mathcal{F}_{\text{sum}}$ (defined in Figure 7), in the presence of a static malicious adversary controlling η fraction of clients and $\eta_{\mathcal{C}}$ fraction of helpers.

E Optimizations

E.1 Sparse LWE

Recall that during the server decryption, it needs to compute $\mathbf{A} \cdot \mathbf{s}$, which is a matrix-vector multiplication. If we use sparse LWE assumption, then most of the entries in $\mathbf{A}$ will be zero, which significantly reduces the time of computing $\mathbf{A} \cdot \mathbf{s}$. The only tradeoff here is security: for a LWE secret of length λ in the standard LWE instance, to guarantee the same level of security in the sparse LWE, we need a secret of length $\lambda' > \lambda$, but concretely only slightly larger [39].

E.2 Client Preprocessing

Since $\mathbf{A}$ is public and the secret $\mathbf{k}_i$ is *independent* of the input, the client can do most of the work of computing $\mathbf{y}_i$ even before it knows the input $\mathbf{x}_i$: once it samples $\mathbf{k}_i$, it computes $\mathbf{A} \cdot \mathbf{k}_i$ and stores it locally. Later when it knows the input $\mathbf{x}_i$, it adds $\mathbf{x}_i$ to the locally stored result together with the error vector. Namely, the online computation only requires a single addition on two vectors of the input length.

E.3 Multi-exponentiation

Naively computing commitments to a length-m vector requires $m+1$ group exponentiations and m group multiplications. We can reduce the number of group exponentiations to sublinear in m using the Pippenger algorithm, given in Lemma 1.

Lemma 1 (Complexity of Pippenger algorithm [56,15,32])**.** Let $\mathbb{G}$ be a group of order $q \approx 2^{\sigma}$, and $G_1, \ldots, G_m$ be m generators of $\mathbb{G}$. Given $v_1, \ldots, v_m \in \mathbb{Z}_q$, Pippenger algorithm can compute $G_1^{v_1} \cdots G_n^{v_m}$ using $\frac{2\sigma m}{\log m}$ group multiplications and σ group exponentiations.

In short, the Pippenger algorithm requires only a small number of expensive group exponentiation (e.g., σ is typically no larger than 256) by increasing the cheap group multiplication by a factor of σ. Therefore, we can use Pippenger for Pedersen vector commitment in any of our inner-product proofs.

E.4 Parameter Selection

We use an LWE security estimator implemented by Albrecht, Player, and Scott [3], which can estimate bits of security when many LWE samples are given (the number of samples equals the vector length in our setting). To have the proof work, we can set the ciphertext modulus q to be the order of the elliptic curve group, such as q being a 253-bit prime for the Ristretto group. However, this requires expensive computation for encryption and decryption. On the other hand, recent efficient protocols based on LWE use machine-friendly q such as 2^{32}. We propose a technique inspired by Angel et al. [5] that does not require setting the ciphertext modulus q equal to the order of the group in the commit-and-proof system.

Proving modulo operation inversely. Let q be the ciphertext modulus in LWE and $Q = |\mathbb{G}|$ where $\mathbb{G}$ is the group for the commit-and-proof system. We set q to be architecture-friendly numbers (e.g., 2^{32}) and keep the group order Q in zero-knowledge proof systems as it is (e.g., a 256-bit prime). Crucially, we require $Q \gg q$ so that wrap-around does not happen (explained next).

The client first performs the encryption *without* any modulo operations. Namely, entries in $\mathbf{A}, \mathbf{k}$ are in $\mathbb{Z}_q$, and we let

$$\widetilde{\mathbf{y}} = \mathbf{A}\mathbf{k} + \mathbf{e} + \lfloor q/p \rfloor \mathbf{x} \in \mathbb{Z}_Q.$$

The ciphertext $\mathbf{y}$ the client sends to the server is $\mathbf{y} \in \mathbb{Z}_q$ where $\widetilde{\mathbf{y}} \mod q$. Note that the client must know a vector $\mathbf{m} \in \mathbb{Z}_Q^{\ell}$ such that

$$\widetilde{\mathbf{y}} = \mathbf{m} \circ (q \cdot \mathbf{1}) + \mathbf{y}.$$

The client will include $\mathbf{m}$ in the witness and prove that

$$\mathbf{y} = \mathbf{A}\mathbf{k} + \mathbf{e} + \lfloor q/p \rfloor \mathbf{x} - \mathbf{m} \circ (q \cdot \mathbf{1}) \quad (1)$$

where the witness consists of $\mathbf{k}, \mathbf{x}, \mathbf{e}, \mathbf{m}$, and $\mathbf{A}, \mathbf{y}$ are public. This technique additionally requires the client to do proof of L_∞ on $\mathbf{x}, \mathbf{e}, \mathbf{k}$ to show their entries are indeed in $\mathbb{Z}_q$, but since the client already did it for $\mathbf{x}, \mathbf{e}$ (see Section 3), the client just additionally does L_∞ proof for $\mathbf{k}$. The same idea can be applied to packed secret sharing as well (the secret $\mathbf{k}$ can be sampled uniformly from $\{0, 1\}$); a caveat is that Shamir secret sharing requires q to be prime (see references [12] for q being power-of-2).

The Schwartz-Zippel optimization can be applied to proving equation 1 even when q does not equal the proof system modulus. Generically, suppose we have $\mathbf{M} \in \mathbb{Z}_q^{W \times L}$ and $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{Z}_q^L$, and we want to prove $\mathbf{v}_2 = \mathbf{M}\mathbf{v}_1 \mod q$. The prover first receives a challenge $r \leftarrow \mathbb{Z}_q$ and let $\mathbf{r} = (r^0, r^1, \ldots, r^{W-1}) \in \mathbb{Z}_q^W$, and it first computes in $\mathbb{Z}_q$ that $\mathbf{a} = \mathbf{M}^\top \mathbf{r} - \mathbf{r}$ and $\mathbf{b} = \mathbf{v}_1 | \mathbf{v}_2$ (entries of $\mathbf{a}, \mathbf{b}$ are in $\mathbb{Z}_q$). We want to prove that $\langle \mathbf{a}, \mathbf{b} \rangle = 0$ in $\mathbb{Z}_q$, but the proof system has $Q \gg q$. To this end, we let the prover find $m \in \mathbb{Z}_Q$ that

$$\langle \mathbf{a}, \mathbf{b} \rangle = q \cdot m \in \mathbb{Z}_Q.$$

Note that $\mathbf{a}$ is a public vector, and $\mathbf{b}$ is a secret vector (witness). Now we can append q to $\mathbf{a}$ (denoted as $\mathbf{a}'$) and append m to $\mathbf{b}$ (denoted as $\mathbf{b}'$) and prove that $\langle \mathbf{a}', \mathbf{b}' \rangle = 0$ using a proof system with modulus Q. Moreover, we additionally need to prove $\|\mathbf{b}\|_\infty < q$, and this is easy to do since $q \ll Q$ (Section C.1). We do not need to prove the L_∞ norm for $\mathbf{a}$ or $\mathbf{a}'$ because they are public.

With the above technique, we can choose the following set of parameters to get over 128-bit security: let λ be 1200, let q be 2^{32}, and let $\mathbf{s}$ and $\mathbf{e}$ be both sampled from normal distribution mod 7, then according to the LWE estimator, this set of parameters gives 129 bits of security and allows $p = 2^{16}$ for summation of 4,000 clients. This should be sufficient for most of the scenarios since the number of clients per iteration varies from 500 to 5K [42, Table 2]).

Vector slicing. We now give the second technique that further reduces the number of helpers. Recall that packed secret sharing allows one to pack λ secrets into a polynomial and each party gets one share. We can instead pack fewer secrets and have each party hold more shares (we call it vector slicing). For example, if $\lambda = 1024$, we can pack $\lambda/8$ secrets (slicing the key vector by a factor of 8) into the polynomial and each client shares 8 polynomials. Now the number of shares held by each helper will be $8n$, but in practice, this is less than 1MB even when $n = 10,000$. When $\lambda = 1024$, we require a total number of helpers only $C = (3/2) \cdot (\lambda/8) = 192$, which is much smaller than n.

A final complication is to incorporate the vector slicing into the zero-knowledge proof design. We can modify our description in Section 3.2 as follows: client i computes vector commitment to each sliced vector of $\mathbf{k}_i$; then it performs the same linear proof as before, except now we have 8 smaller instances. The client sends to the server the vector commitments to the 8 sliced vectors which allows the server to compute the commitment to $\mathbf{k}_i$.

F Cost Overview of Armadillo

To better understand the concrete cost, we first give a cost overview of the client and the server.

Cost overview of Armadillo. A client first masks its input vector of length m using the preprocessed vector (Section E.2); this involves m additions over $\mathbb{Z}_q$. Also, the client computes C shares of the key and the corresponding C Pedersen commitments to the shares; this is dominated by $2C$ group exponentiations. The client then creates the proofs, of which the cost is stated in Theorem 1.

Each helper j receives and verifies n Pedersen commitments, which is $2n$ group exponentiations. It kicks out the clients whose shares fail the verification. Then the helper adds up the valid shares and sends the sum to the server; this is at most n additions on field elements.

Baselines. The most related work is ACORN-robust [9], where they provide the same input validation guarantee as our protocol but achieve robustness in a very different way. We provide details in Appendix G.2. Roughly, ACORN-robust follows the pairwise masking approach by Bell et al. [8], and each client does the proof of input validity same as ours, but additionally computes Feldman commitments to the shares of its pairwise secrets, which is later useful for identifying cheating clients.

The other protocols that achieve input validation (but without robustness) are Eiffel [24], and RoFL [50]. It was shown in [9] that Eiffel and RoFL are both more expensive than ACORN, so we do not use them as baselines here.

Libraries, testbed, and parameters. We implement our protocol using Rust. For instantiating the proofs, we use Nova [60,44,2], which is an R1CS-based proof system. For linear and quadratic proof, this has faster prover and verifier time compared to the state-of-the-art rust implementation of Bulletproof [28]. We run our experiment on a laptop with a 2.4GHz Apple M2 chip. The range of the clients' inputs are integers in $[0, 2^{16} - 1]$. Both the client and server-side experiments vary the length of the inputs from 2^{11} to 2^{15}. Note that 2^{10} is too small for our protocol to make sense since the LWE secret already has dimension on par with 2^{10}.

G Details on Baseline Protocols

G.1 Cost Overview of ACORN-detect

To understand how ACORN-robust works we first present ACORN-detect. In ACORN-detect protocol, the server can detect if a client cheats but the protocol does not have guaranteed output delivery. We outline the protocol below and briefly analyze its cost.

We start with the protocol (without input validation) in Bell et al. [8] which is also a base protocol for ACORN. Initially, the server establishes a public graph on all n clients where each client has $k = O(\log n)$ neighbors; let $N(i) \subset [n]$ denote the neighbors of i. Each pair of clients establish pairwise secrets p_{ij}. Each client i generates a random PRG seed z_i and masks the input $\mathbf{x}_i$ as

$$\mathbf{y}_i = \mathbf{x}_i + \mathbf{r}_i,$$

where the mask $\mathbf{r}_i$ is defined as

$$\mathbf{r}_i = \sum_{i<j, j\in N(i)} \mathtt{PRG}(p_{ij}) - \sum_{i>j, j\in N(i)} \mathtt{PRG}(p_{ij}) + \mathtt{PRG}(z_i).$$

The client sends $\mathbf{y}_i$ to the server. Note that z_i is for ensuring privacy when handling dropouts; see more details in Bell et al. [8]. We skip the rest of details of the protocol here, but their key feature is that for any online set $\mathcal{O} \subset [n]$, the server eventually gets $\mathbf{r} := \sum_{i\in\mathcal{O}} \mathbf{r}_i$ so that it can remove $\mathbf{r}$ from $\mathbf{y} := \sum_{i\in\mathcal{O}} \mathbf{y}_i$ and obtain the desired output $\sum_{i\in\mathcal{O}} \mathbf{x}_i$.

To achieve input validity, they added the following steps to the above protocol. Each client i computes the commitment to $\mathbf{x}_i$ and the commitment to the aggregated mask $\mathbf{r}_i$ and sends them together with the masked vector $\mathbf{y}_i = \mathbf{x}_i + \mathbf{r}_i$. Then the client proves that

- $\mathbf{x}_i$ has valid L_2, L_∞ norm (same as Section C.1);
- It added $\mathbf{r}_i$ to $\mathbf{x}_i$ correctly (which can be done using a linear proof).

Recall that the server learns $\mathbf{r} := \sum_{i\in\mathcal{O}} \mathbf{r}_i$. Next, the clients and the server run a distributed key correctness (DKC) protocol to check if the server obtains $\mathbf{r} := \sum_{i\in\mathcal{O}} \mathbf{r}_i$ where the $\mathbf{r}_i$'s are indeed consistent with the commitments that the clients sent in the first place.

Remark 1. When the PRG is instantiated with homomorphic PRG (e.g., RLWE-based PRG), the client can optimize its computation by first computing the sum of the seeds and then expanding the aggregated seeds with PRG. A trade-off is that the masking here is not simply $\mathbf{x}_i + \mathbf{r}_i$: since the PRG output is defined over polynomial rings, the input $\mathbf{x}_i$ should be interpreted as polynomials when added to $\mathbf{r}_i$ and this requires non-trivial encoding of $\mathbf{x}_i$ (see Equation 5 in ACORN [9]). As a result, the client also needs to prove it performs the encoding correctly.

Cost. Each client computes two vector commitments to length ℓ vectors $\mathbf{x}_i, \mathbf{r}_i$. For the DKC protocol, the client performs a constant number of elliptic curve scalar multiplications, and the server performs $3n$ of them.

G.2 Cost Overview of ACORN-robust

ACORN-robust is similar to ACORN-detect but with the following differences:

- The pairwise secrets are established differently (see details below);
- When the server fails verification in the DKC protocol, it invokes an $O(\log n)$-round bad message resolution protocol with all the clients to remove the malicious clients' contribution from the sum.

Suppose the server establishes a public graph on all n clients where each client has $k = O(\log n)$ neighbors. First, each client i generates k seeds $s_{i,j}$ for neighbor j, and sends them to the neighbors; client i additionally generates (deterministic) commitments to the seeds, namely $s_{i,j} \cdot G$, which are sent to the server. Next, clients exchange the seeds with their neighbors: a client i neighboring with client j will send $s_{i,j}$ and receive $s_{j,i}$, and vice versa. Client i and j then establish pairwise secret $p_{ij} = s_{i,j} + s_{j,i}$; this p_{ij} will be used for pairwise-masking the input vector.

Each client i then Shamir-shares $s_{i,j}$ and sends the Feldman commitments to the sharing of $s_{i,j}$ (commitments to the coefficients of the sharing polynomial) to the server. The server checks if the

Functionality $\mathcal{F}$

Parties: A set of n clients $P_1, \ldots, P_n$ and a server S.
Parameters: corruption rate η and dropout rate δ among $\{P_1, \ldots, P_n\}$.
Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ and $\mathcal{X} = \mathcal{P} \cup \mathcal{S}$.

- $\mathcal{F}$ receives from the adversary a set of corrupted parties $\mathcal{A} \subset \mathcal{X}$, where $|\mathcal{A} \cap \mathcal{P}| \leq \eta n$.
- $\mathcal{F}$ receives a set of dropout clients $\mathcal{O} \subset \mathcal{P}$, and inputs $\mathbf{x}_i$ for client $P_i \in \mathcal{P} \backslash (\mathcal{O} \cup \mathcal{A})$.
- The output of $\mathcal{F}$:
  1. If $S \notin \mathcal{A}$, then $\mathcal{F}$ outputs $\mathbf{z} = \sum_{P_i \in \mathcal{P} \backslash (\mathcal{A} \cup \mathcal{O})} \mathbf{x}_i$;
  2. If $S \in \mathcal{A}$, then $\mathcal{F}$ asks the adversary for a set: if the adversary replies with a set $\mathcal{M} \subseteq \mathcal{P}$ where $|\mathcal{M}| \geq (1-\delta)n$, then $\mathcal{F}$ outputs $\mathbf{z}' = \sum_{P_i \in \mathcal{M} \backslash (\mathcal{A} \cup \mathcal{O})} \mathbf{x}_i$; otherwise, $\mathcal{F}$ sends `abort` to clients $P_i \in \mathcal{P} \backslash \mathcal{A}$.

Figure 7: Ideal functionality of a private sum with robustness. We follow the definition in prior works [13,8] that assumes an oracle gives a valid dropout set to $\mathcal{F}$.

Feldman commitments match the commitment $s_{i,j} \cdot G$. If not, the server disqualifies client i; if it matches, the server computes $s_{i,j}^{(k)} \cdot G$ from Feldman commitments, where $s_{i,j}^{(k)}$ is the share meant for the k-th neighbor of client i. Then the server sends $s_{i,j}^{(k)} \cdot G$ to the corresponding client. The recipient client checks if the decrypted share $s_{i,j}^{(k)}$ matches the commitment $s_{i,j}^{(k)} \cdot G$. Then the server and clients invoke an $O(\log n)$-round bad message resolution protocol to form a set of clients whose pairwise masks can be canceled out.

There are two costly parts of ACORN-robust: 1) the obvious complexity of the logarithmic number of rounds between the server and all the clients; 2) the server needs to verify $O(n \log n)$ Feldman commitments of sharing of degree $\log n$. Concretely, using the parameters estimated by Bell et al., with $n = 1000$ clients and $\delta = \eta = 0.05$, the neighbors required (for security) is roughly 30, meaning that here the server needs to perform $2 \cdot 30^2 \cdot 1000 = 1,800,000$ elliptic curve scalar multiplications and this takes roughly 10 minutes; note that this cannot be trivially optimized with multi-exponentiation because the server needs to identify the malicious clients.

H Security proof of Armadillo

H.1 Proof of Theorem 2

The proof for dropout resilience is simple. When all parties follow the protocol, given a dropout set $\mathcal{O} \subset [n]$, the server will compute $\mathbf{y} = \sum_{i \in [n] \backslash \mathcal{O}} \mathbf{y}_i$ (Round 4 in Fig.5). For the inner aggregation, the set of helpers $\mathcal{C}$ will obtain shares of $\mathbf{k}_i$ for all $i \in [n] \backslash \mathcal{O}$ (because the server honestly forwards the messages). Assuming $d < C(1 - \delta_{\mathcal{C}})$, the server will get the result from inner aggregation which is $\mathbf{k} = \sum_{i \in [n] \backslash \mathcal{O}} \mathbf{k}_i$. Therefore, we have $\mathsf{Dec}(\mathbf{k}, \mathbf{y}) = \sum_{i \in [n] \backslash \mathcal{O}} \mathbf{x}_i$.

H.2 Proof of Theorem 3

We follow the proof of security similar to that of ACORN-robust [9]. However, there are key differences. Their protocol guarantees the privacy of honest clients only with a semi-honest server. This is an artifact of their protocol where the server is empowered to recover the masks—both the self-masks and pairwise masks—for misbehaving clients to then remove their inputs. In other words, the server is capable of recovering the actual inputs of malicious clients. Consequently, a malicious server could claim honest clients to be malicious and thereby recover the inputs of these clients. In contrast, our protocol works by using a single mask, and these masks are never revealed to the server, even for those misbehaving clients.

Our proof methodology relies on the standard simulation-based proof, where we show that every adversary attacking our protocol can be simulated by an adversary Sim in an ideal world where the functionality $\mathcal{F}$ (Fig.7). In the following, we first prove privacy against any adversary corrupting ηn

clients and the server; then we prove robustness assuming the adversary corrupting ηn clients but not the server (recall our threat model in §1.1).

The challenge in the simulation is the ability of Sim to generate a valid distribution for the honest clients' inputs, even without knowing their keys. To this end, we will show that Sim, when only given the sum of the user inputs $\mathbf{X} = \sum_{i=1}^{n} \mathbf{x}_i$, can simulate the expected leakage for the server which includes n ciphertexts, the sum of the n keys $\mathbf{K} = \sum_{i=1}^{n} \mathbf{k}_i$, and such that the sum of the n ciphertexts, when decrypted with $\mathbf{K}$, correctly decrypts to $\mathbf{X}$.

Before we detail the definition of Sim and prove its security, we present an assumption that we will use later.

Definition 1 (A variant of Hint-LWE [47,23])**.** Consider integers λ, m, q and a probability distribution χ' on $\mathbb{Z}_q$, typically taken to be a normal distribution that has been discretized. Then, the Hint-LWE assumption states that for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that:

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m\times\lambda}, \mathbf{k} \xleftarrow{\$} \mathbb{Z}_q^{\lambda}, \mathbf{e} \xleftarrow{\$} \chi'^m \\ \mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^{\lambda}, \mathbf{f} \xleftarrow{\$} \chi'^m \\ \mathbf{y}_0 := \mathbf{A}\mathbf{k} + \mathbf{e}, \mathbf{y}_1 \xleftarrow{\$} \mathbb{Z}_q^m, b \xleftarrow{\$} \{0,1\} \\ b' \xleftarrow{\$} \mathcal{A}(\mathbf{A}, (\mathbf{y}_b, \mathbf{k}+\mathbf{r}, \mathbf{e}+\mathbf{f})) \end{array} \right] = \frac{1}{2} + \mathsf{negl}(\kappa)$$

where κ is the security parameter.

Intuitively, Hint-LWE assumption says that $\mathbf{y}_0$ looks pseudorandom to an adversary, even when given some randomized leakage on the secret and the error vectors. Kim et al. [43] show that solving Hint-LWE is no easier than solving LWE problem. For a secure LWE instance (λ, m, q, χ) where χ is a discrete Gaussian distribution with standard deviation σ, the corresponding Hint-LWE instance (λ, m, q, χ'), where χ' is a discrete Gaussian distribution with standard deviation σ', is secure when $\sigma' = \sigma/\sqrt{2}$. Consequently, any $\mathbf{e} \in \chi$ can be written as $\mathbf{e}_1 + \mathbf{e}_2$ where $\mathbf{e}_1, \mathbf{e}_2 \in \chi'$. This gives us the real distribution $\mathcal{D}_R$, with the error term re-written and the last ciphertext modified.

$$\left\{ \begin{array}{c} \mathbf{K} = \sum_{i=1}^{n} \mathbf{k}_i \bmod q \\ \mathbf{y}_1, \ldots, \mathbf{y}_n \\ \end{array} \;\middle|\; \begin{array}{c} \forall i \in [n], \mathbf{k}_i \xleftarrow{\$} \mathbb{Z}_q^{\lambda}, \mathbf{e}_i, \mathbf{f}_i \xleftarrow{\$} \chi'^m \\ \forall i \in [n-1], \mathbf{y}_i = \mathbf{A} \cdot \mathbf{k}_i + \mathbf{e}_i + \Delta \mathbf{x}_i \\ \mathbf{y}_n = \mathbf{A}\mathbf{K} - \sum_{i=1}^{n-1} \mathbf{y}_i + \sum_{i=1}^{n} (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{X} \end{array} \right\}$$

We now define $\mathsf{Sim}(\mathbf{A}, \mathbf{X})$:

$\underline{\mathsf{Sim}(\mathbf{A}, \mathbf{X})}$

Sample $\mathbf{u}_1, \ldots, \mathbf{u}_{n-1} \xleftarrow{\$} \mathbb{Z}_q^m$

Sample $\mathbf{k}_1, \ldots, \mathbf{k}_n \xleftarrow{\$} \mathbb{Z}_q^{\lambda}$

Sample $\mathbf{e}_1, \ldots, \mathbf{e}_n \xleftarrow{\$} \chi'^m$

Sample $\mathbf{f}_1, \ldots, \mathbf{f}_n \xleftarrow{\$} \chi'^m$

Set $\mathbf{K} := \sum_{i=1}^{n} \mathbf{k}_i \bmod q$

Set $\mathbf{u}_n = \mathbf{A} \cdot \mathbf{K} - \sum_{i=1}^{n-1} \mathbf{u}_i + \sum_{i=1}^{n} (\mathbf{e}_i + \mathbf{f}_i) + \Delta \cdot \mathbf{X}$

Return $\mathbf{K}, \mathbf{u}_1, \ldots, \mathbf{u}_n$

In other words, the simulated distribution, $\mathcal{D}_{\mathsf{Sim}}$, is:

$$\left\{ \begin{array}{c} \mathbf{K} = \sum_{i=1}^{n} \mathbf{k}_i \bmod q \\ \mathbf{u}_1, \ldots, \mathbf{u}_n \\ \end{array} \;\middle|\; \begin{array}{c} \forall\, i \in [n]\; \mathbf{k}_i \xleftarrow{\$} \mathbb{Z}_q^{\lambda}, \mathbf{e}_i, \mathbf{f}_i \xleftarrow{\$} \chi'^m \\ \forall\, i \in [n-1]\; \mathbf{u}_i \xleftarrow{\$} \mathbb{Z}_q^m \\ \mathbf{u}_n = \mathbf{A}\mathbf{K} - \sum_{i=1}^{n-1} \mathbf{u}_i + \sum_{i=1}^{n} (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{X} \end{array} \right\}$$

We will now prove that $\mathcal{D}_R$ is indistinguishable from $\mathcal{D}_{\mathsf{Sim}}$ through a sequence of hybrids.

- Hybrid 0: This is $\mathcal{D}_R$.

- Hybrid 1: In this hybrid, we will replace the real ciphertext $\mathbf{y}_1$ with a modified one. In other words, we set:

$$\left\{ \begin{array}{c|c} \mathbf{K} & \forall\, i \in [n]\ \mathbf{k}_i \xleftarrow{\$} \mathbb{Z}_q^{\lambda}, \mathbf{e}_i, \mathbf{f}_i \xleftarrow{\$} \chi'^m, \mathbf{u}'_1 \xleftarrow{\$} \mathbb{Z}_q^m \\ \mathbf{y}_1 = \mathbf{u}'_1 + \mathbf{f}_1 + \Delta\mathbf{x}_1 & \forall\, i \in [2, n-1]\ \mathbf{y}_i = \mathbf{A} \cdot \mathbf{k}_i + (\mathbf{e}_i + \mathbf{f}_i) + \Delta\mathbf{x}_i \\ \{\mathbf{y}_i\}_{i=2}^n & \mathbf{y}_n = \mathbf{A}\mathbf{K} - \sum_{i=1}^{n-1} \mathbf{y}_i + \sum_{i=1}^{n} (\mathbf{e}_i + \mathbf{f}_i) + \Delta\mathbf{X} \end{array} \right\}$$

Now, we will show that if there exists an adversary $\mathcal{B}$ that can distinguish between Hybrid 0 and 1, then we can define an adversary $\mathcal{A}$ who can distinguish the two ensembles in the Hint-LWE Assumption. Let us define $\mathcal{A}$ now.

$\underline{\mathcal{A}(\mathbf{A}, \mathbf{y}^*, \mathbf{k}^* = \mathbf{k} + \mathbf{r} \bmod q, \mathbf{e}^* = \mathbf{e} + \mathbf{f})}$

Sample $\mathbf{k}_2, \ldots, \mathbf{k}_{n-1} \xleftarrow{\$} \mathbb{Z}_q^{\lambda}$
Sample $\mathbf{e}_2, \ldots, \mathbf{e}_n \xleftarrow{\$} \chi'^m$
Sample $\mathbf{f}_2, \ldots, \mathbf{f}_n \xleftarrow{\$} \chi'^m$
Set $\mathbf{K} = \sum_{i=2}^{n-1} \mathbf{k}_i + \mathbf{k}^* \bmod q$ // implicitly, $\mathbf{k}_n := \mathbf{r}$
$\forall\, i \in \{2, \ldots, n-1\},\ \mathbf{y}_i = \mathbf{A}\mathbf{k}_i + \mathbf{e}_i + \mathbf{f}_i + \Delta\mathbf{x}_i$
Set $\mathbf{y}_1 = \mathbf{y}^* + \mathbf{f}_n + \Delta\mathbf{x}_1$
Set $\mathbf{y}_n := \mathbf{A}\mathbf{K} - \sum_{i=1}^{n-1} \mathbf{y}_i + \mathbf{e}^* + \sum_{i=2}^{n} (\mathbf{e}_i + \mathbf{f}_i) + \Delta \cdot \mathbf{X}$
Run $b' \xleftarrow{\$} \mathcal{B}(\mathbf{K}, \mathbf{y}_1, \ldots, \mathbf{y}_n)$
return b'

We need to argue that the reduction correctly simulates the two hybrids, based on the choice of y^*.

  – If $\mathbf{y}^* = \mathbf{A}\mathbf{k} + \mathbf{e}$, then $\mathbf{y}_1$ is a valid encryption of $\mathbf{x}_1$ with key $\mathbf{k}$ and error $(\mathbf{e} + \mathbf{f}_n)$. Further, it is easy to verify that $\mathbf{y}_n$ satisfies the definition present in Hybrid 0.
  – If $\mathbf{y}^* = \mathbf{u}$ for some random $\mathbf{u}$. Then, we get that $\mathbf{y}_n$ is of the prescribed format, while also guaranteeing that $\mathbf{y}_1$ is generated as expected.

- Hybrid 2: In this hybrid, we will replace $\mathbf{y}_1$ with $\mathbf{y}_1$ that is sampled uniformly at random.

$$\left\{ \begin{array}{c|c} \mathbf{K} & \forall\, i \in [n]\ \mathbf{k}_i \xleftarrow{\$} \mathbb{Z}_q^{\lambda}, \mathbf{e}_i, \mathbf{f}_i \xleftarrow{\$} \chi'^m, \mathbf{u}_1 \xleftarrow{\$} \mathbb{Z}_q^m \\ \mathbf{u}_1 & \forall\, i \in [2, n-1]\ \mathbf{y}_i = \mathbf{A} \cdot \mathbf{k}_i + (\mathbf{e}_i + \mathbf{f}_i) + \Delta\mathbf{x}_i \\ \{\mathbf{y}_i\}_{i=2}^n & \mathbf{y}_n = \mathbf{A}\mathbf{K} - \mathbf{u}_1 - \sum_{i=2}^{n-1} \mathbf{y}_i + \sum_{i=1}^{n} (\mathbf{e}_i + \mathbf{f}_i) + \Delta\mathbf{X} \end{array} \right\}$$

Hybrid 1, and Hybrid 2 are identically distributed $\mathbf{u}'_1$ is uniformly sampled and essentially mask the values in $\mathbf{y}_1$ of Hybrid 1.

In Hybrids 3 and 4, we replace $\mathbf{y}_2$ with a random element $\mathbf{u}_2$, by using a similar logic. Therefore, in Hybrid $2n-2$, the distribution will resemble $\mathcal{D}_{\mathsf{Sim}}$. This concludes the proof of simulatability.

Privacy. Here we prove privacy against an attacker corrupting the server and a set of ηn clients (some of them can be helpers). Denote the simulator as Sim_p. The formal proof proceeds through a sequence of hybrids. The sequence of hybrids is similar to the work of Bell et al. [8]. Let $\mathcal{H} = [n] \setminus \mathcal{C}$. Below, we detail the hybrids.

- Hybrid 0: This is the real execution of the protocol where the adversary is interacting with honest parties.
- Hybrid 1: This is where we introduce a simulator Sim which knows all the inputs and secret keys involved, i.e., it knows the keys and the shares of all the clients. Sim runs a full execution of the protocol with the adversary and programs the random oracle as needed. The view of the adversary in this hybrid is indistinguishable from the previous hybrid.
- Hybrid 2: Our next step is for the simulator Sim to rely on the Special Honest Verifier Zero Knowledge (SHVZK) property of all the proof systems to simulate the zero-knowledge proofs for each honest client. Any non-negligible distinguishing advantage between Hybrids 1 and 2 will violate the SHVZK property of the underlying proof systems.

- Hybrid 3: In the next step, we rely on the hiding property of Pedersen commitments. Recall that the hiding property guarantees that there is a negligible distinguishing advantage for an adversary between an actual Pedersen commitment and a random group element. Therefore, for all the honest clients, Sim can simply replace the commitments provided with a random group element. Any non-negligible distinguishing advantage between Hybrids 2 and 3 will violate the hiding property of the commitment scheme.
- Hybrid 4: In the next step, we rely on the privacy property of Shamir Secret Sharing. This guarantees that any insufficient number of shares does not leak the privacy of the secret. In this hybrid Sim uses this property to replace the shares of the honest user's keys meant for the corrupt helpers with random values. Recall that the number of corrupt helpers is strictly less than the reconstruction threshold. Therefore, any non-negligible advantage in distinguishing advantage between Hybrids 3 and 4 will imply that the statistical security of Shamir's Secret Sharing is broken.

  Thus far, for the honest clients' Sim has successfully generated all the contributions for the honest users, except for the ciphertexts themselves. However, Sim cannot simply rely on the semantic security of LWE encryption to replace with encryptions of random values. This is because the output might differ from the real world. Instead, Sim, which has control of the corrupted parties, simply instructs the corrupted parties to provide their inputs as $\mathbf{0}$. Then, the output of the functionality is simply the sum of the honest clients' inputs. Let us call it $\mathbf{x}_H$. With this knowledge, Sim can generate its own choices of individual inputs for honest clients, with the only constraint that the values necessarily need to sum up $\mathbf{x}_H$. This guarantees that the output is correct.
- Hybrid 5: Sim now relies on the semantic security of LWE encryption, under leakage resilience as argued earlier in this section, to instead encrypt these sampled values for honest clients. Any non-negligible distinguishing advantage between Hybrids 4 and 5 will imply that the LWE encryption is no longer semantically secure.

At Hybrid 5, it is clear that Sim can successfully simulate a valid distribution that does not rely on the honest party's inputs. This concludes the proof.

Remark 2 (On privacy of ACORN-robust)**.** A critical artifact of ACORN-robust in [9] is the loop-based resolution of malicious behavior. Specifically, the protocol relies on a looping process by which the server identifies some malicious clients in every round of communication. This is done by finding inconsistencies in the clients' communication. Unfortunately, once a misbehaving client is detected, the protocol necessarily needs to communicate with the parties to retrieve the self-mask *and* the pairwise masks along each edge of the neighborhood graph. Consequently, the server receives all the information necessary to unmask the inputs. Therefore, a malicious server could conceivably claim an honest client to be a misbehaving client, thereby compromising the privacy of the inputs. This is acknowledged by the authors of [9]. However, a simple fix would be for the server to attach necessary proofs of malicious behavior but the communication involved in this process is higher.

(a) the honest clients send their inputs to T , (b) A chooses which corrupted clients send their input to T and which ones abort, (c) if the server is corrupted A gets to choose whether to abort the protocol or continue, and (d) if the protocol is not aborted, T gives the server its prescribed output F (X). Finally, (e) if the server is not corrupted then it outputs what it received from T .

Robustness. Now we turn to proving robustness (and also showing privacy) when the adversary corrupting only a set of ηn clients (some of them can be helpers). Here the server follows the protocol, but can try to violate the privacy.

We denote the simulator here as Sim_r. Note that in the ideal world Sim_r has to provide the inputs for both the honest and corrupted clients. Meanwhile, in the real world the inputs for the corrupted clients comes from the adversary, call it $\mathcal{B}$. Note that $\mathcal{B}$ can choose these inputs, with any restrictions of its own. Therefore, to ensure that it produces a valid set of inputs to the functionality in the ideal world, Sim_r does the following:

- It invokes $\mathcal{B}$ by internally running it. Sim_r honestly follows the protocol, fixing the inputs for the honest clients to be some valid vector $\mathbf{X}$. To $\mathcal{B}$, this is an expected run and therefore it behaves exactly like in the real world execution.

- Sim_r records the set of corrupted parties $\mathcal{A}$ and the set of dropout clients $\mathcal{O}$ encountered in this internal execution.
- At some point, $\mathcal{B}$ provides the NIZK proofs to the server for adversarial clients. However, Sim_r controls the server with these proofs including proof of Shamir sharing, proof of correct encryption, range proofs, and the proof of binding of shares and the key.
- Using the Knowledge Soundness property of the NIZK proofs, Sim_r is able to extract the witnesses, specifically the inputs for the adversarial clients.
- Finally, Sim_r also records whatever $\mathcal{B}$ outputs in the internal execution.

With these steps in place, Sim_r can simulate the ideal world.

- It sends the recorded $\mathcal{O}, \mathcal{A}$ to the ideal functionality.
- It sends the extracted adversarial inputs for those clients, while sending the valid inputs for the non-dropout honest clients.
- Note that the inputs in both the real-world and ideal-world match. We need to show that the computed output matches too.
- Finally, Sim_r outputs whatever $\mathcal{B}$ had output in the internal execution.

It is clear that the output of Sim_r (in the ideal world) is indistinguishable from the output of $\mathcal{B}$ (in the real world). However, we now need to argue that the output sum cannot differ at all. Specifically, while it is guaranteed that the adversarial inputs are included in the sum in the real world (as it was done in the internal execution of $\mathcal{B}$). We need to show that the honest clients' inputs cannot be dropped from the computed sum.

To see this, observe that the server only removes a client if there is a proof of the client misbehaving. As a corollary, it implies that an honest party's input is never rejected by the honest server as it would not have proof of malicious behavior. This guarantees that any honest client's inputs, which hasn't dropped out, is always included in the computed sum in the real world. In other words, the computed sum in the real and ideal world have to match.

H.3 A Fix to ACORN-detect

We clarify details related to counting rounds in experiments, point out an overlooked issue in ACORN-detect, and propose a patch.

ACORN-detect, as described in Figure 6 of [9], achieves input validation by integrating the distributed key correctness (DKC) protocol (as described in Figure 2 of [9]) and zero-knowledge proof into the main secure aggregation protocol. The DKC protocol is an interactive protocol which helps the server verify that the masks the server reconstructs is what the clients committed to when sending the masked inputs to the server. In the protocol description of ACORN-detect in Figure 6, communication of the distributed key correctness protocol is embedded into the main protocol, thus there is no additional communication round incurred. However, it seems that the authors overlooked the assumption that the clients can drop offline in any round in the protocol execution when plugging the DKC protocol into ACORN-detect. More specifically, the set of clients who participate in step 8 in ACORN-detect (which contains step 3 of DKC) in which each client sends both the masked input and the commitment to the mask the user might be a superset of the set of clients participating in Step 10 (which contains step 5 of DKC) in which each client sends the server the information needed to verify the commitment of the mask if some clients drop offline between these two rounds. Note that the set $\mathcal{O}$ of clients whose inputs are chosen to be included in the final result is determined when the server receives the masked input in step 9 of ACORN-detect and is not changed later. As a consequence, in the last step of ACORN-detect (which contains step 6 of DKC), the server is not able to collect all the information needed for the key verification for the online set and the server will abort due to the verification failure even when all participants are honest, which breaks dropout resilience. This problem can be fixed by extracting step 4 and 5 of the DKC protocol from ACORN-detect as a separate round between steps 8 and 9 of ACORN-detect rather than embedded in step 9 and 10 of ACORN-detect and determining the online set $\mathcal{O}$ by who sends both the commitment of the masks and the information needed for the verification of the commitment. This fix introduces one extra round to ACORN-detect.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS paper checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The paper sets out to solve a critical problem in prior work on secure aggregation. In this work, we demonstrate how to guarantee robustness of model, in the face of malicious clients by identifying and removing these bad actors. We demonstrate experiments to show competitive performance over prior work.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: We have a conclusion paragraph that draws attention to some of the limitations while identifying how they can be remedied in future work.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper introduces all the necessary theoretical framework and assumptions for security of the construction. There's detailed proof deferred to the appendix.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The protocols are well detailed, including the parameter settings for our classifier. We use publicly available ABIDES framework to simulate real-life networking situations.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [NA]

Justification: Unfortunately, there was no support for supplementary material upload. However, we are happy to furnish the anonymized code for interested reviewers.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [NA]

   Justification: We do not perform any training or testing in this work.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification: Our running time experiments report the mean as specified.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We detail the system settings of the device on which experiments are performed.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The paper complies with the code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The work focuses on privacy of client-held data. This is surveyed in the introduction and motivates why privacy-preserving federated learning is important and its positive impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: We do not use any of the stated models or data sources.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [NA]

Justification: This is not applicable for our work.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: We are not releasing any new assets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: There were no human subjects involved in this project.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: We do not have any research with human subjects that forms a part of this work.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.