Algorithm 2 Federated learning algorithm [2, 1]

Initialize global model θ_a^0
for $t \in [T]$ **do**
  for $i \in [n]$ **do** ▷ Party i's local update
    $\theta_i^t \leftarrow \theta_a^t - \nabla_\theta L(D_i; \theta_a^t)$
    Return θ_i^t to server
  end for
  $\theta_a^{t+1} = f_{\mathsf{AGG}}(\theta_{i\in[n]}^t)$ ▷ Aggregation of updates at server
  Return θ_a^{t+1} to all the parties
end for

A Aggregation algorithms

In this section, we describe the setting of federated learning and, various aggregation algorithms used to combine the updates of the collaborating parties.

A.1 Federated learning setting

Federated learning [2, 1] enables multiple data holders to train a global model without sharing their data, and through sharing of their training gradients/parameters [10, 45, 8, 46, 9]. For concreteness, below we describe Federated Average (FedAvg) algorithm [2] and its setting, and use it in the rest of our work. In FedAvg, multiple parties collaborate over multiple epochs to learn a global machine learning model with a classification performance superior to the models learned individually. FedAvg assumes that there are n parties with their *local training datasets*, D_i's, and a *central server* which aggregates the party updates and broadcasts the aggregate to all of the parties. In the t^{th} epoch of FedAvg, parties train the aggregate θ_a^t, broadcast by the server at the end of t^{th} epoch, on their local training data, D_i. Parties use stochastic gradient descent for updating, i.e., $\theta_i^t = \theta_a^t - \nabla_\theta L(D_i; \theta_a^t)$, where $L(D; \theta)$ is loss of θ on data D. Each party then sends the parameters of the locally updated model, θ_i^t, to the server for aggregation. The central server collects all the θ_i^t updates and computes their aggregate $\theta_a^t = f_{\mathsf{AGG}}(\theta_{i\in[n]}^t)$. Specifically, FedAvg uses the weighted average as its f_{AGG}, where the weight of the i^{th} party is determined based on the size of her local training data, i.e., $w_i = \frac{|D_i|}{|D|}$; $|D|$ denotes size of dataset D. The weighted average is formally given by:

$$f_{\mathsf{Mean}}: \quad \theta_a^{t+1} = \sum_{i=1}^{n} \frac{|D_i|}{\sum_{j=1}^{n} |D_j|} \theta_i^t \tag{1}$$

The server then broadcasts the aggregate θ_a^t to all n parties. This process repeats for T epochs or until sufficient accuracy is achieved by the aggregated global model. The procedure is described in Algorithm 2. This algorithm is not robust against poisoning attack and even a single party can destroy the global model.

A.2 Krum

Weighted average aggregation cannot tolerate even a single malicious party [10, 47]. To solve this problem, Blanchard et al. [10] proposed Krum aggregation, which is based on geometric median of vectors [48, 8]. The intuition behind Krum is as follows: Krum assumes that party updates have a normal distribution and that the benign updates lie close to each other in the parameter space. Hence, instead of computing the average of the updates, Krum selects as aggregate the update that is closest to its $(1-\epsilon)n$ neighbor updates. The details of the aggregation are as follows. Let $\theta_1, ..., \theta_n$ be the updates received by the server. For $i \neq j$, $i \to j$ denotes that θ_j belongs to the $(1-\epsilon)n - 2$ updates closest to θ_i. Let $s(\theta_i) = \sum_{i \to j} ||\theta_i - \theta_j||^2$ be the *score* of θ_i. Then, Krum selects the θ_k with the lowest score. The Krum aggregation algorithm is formalized in (2).

$$f_{\mathsf{Krum}}: \quad \theta_a^{t+1} = \underset{\theta_{i\in[n]}^t}{\operatorname{argmin}} \sum_{i \to j} ||\theta_i^t - \theta_j^t||^2 \tag{2}$$

A.3 Bulyan

The breaking point of Krum is $\epsilon = (\frac{n-2}{2n})$, i.e., it can tolerate up to $(\frac{n-2}{2})$ malicious parties, while maintaining high utility of the final model [10]. However, El Mhamdi et al. [8] proposed an attack on Krum assuming an omniscient adversary who has access to all the benign updates. The attack exploits the fact that, in a vector space of dimension $d \gg 1$, small disagreements on each coordinate translate into a distance $\|\mathbf{x} - \mathbf{y}\|_p = \mathcal{O}(\sqrt[p]{d})$. Therefore, the adversary crafts a malicious update with a single dimension set to a large value, and the other dimensions set to the average of the benign updates. Such malicious update pushes the parameter vector to a sub–optimal parameter space and destroys the global model's accuracy. Essentially, Krum filters outliers based on the entire update vector, but does not filter coordinate-wise outliers. To address this, [8] proposes a meta-aggregation rule Bulyan, which performs vector-wise, e.g. Krum, and coordinate-wise, e.g. TrimmedMean [9], filtering in two steps. At first, Bulyan uses some Byzantine resilient aggregation $\mathcal{A}$, e.g., Krum in Algorithm 3, to filter outliers based on the distances between the update vectors, and then aggregates these updates using a variant of TrimmedMean. Algorithm 3 describes the Bulyan aggregation.

Algorithm 3 Bulyan aggregation: f_{Bulyan} [8]

Input: $\mathcal{A} = f_{\mathsf{Krum}}$, $\mathcal{P} = (\theta_1^t, ..., \theta_n^t)$, n, ϵ
$S \leftarrow \emptyset$
while $|S| < (1 - 2\epsilon)n$ **do**
  $p \leftarrow \mathcal{A}(\mathcal{P} \backslash S)$
  $S \leftarrow S \cup \{p\}$
end while
Output: $\theta_a^{t+1} = \mathsf{TrimmedMean}(S)$

Among the different variants of TrimmedMean [46, 8, 9], we follow the one used in the original work [8] given in (3). Here, U_j is defined as the set of indices of the top-$(1 - 2\epsilon)n$ values in $(\theta_1^j, ..., \theta_n^j)$ nearest to their *median* μ_j.

$$\mathsf{TrimmedMean}(\theta_1, ..., \theta_N) = \left\{ \theta_a^j = \frac{1}{|U_j|} \sum_{i \in U^j} \theta_i^j \ \forall j \in [d] \right\} \tag{3}$$

A.4 Multiplicative weight update (MWU)

Multiplicative weight update (MWU) technique lies at the core of many learning algorithms [24, 27, 28, 29, 25]. The general framework of MWU is given in Algorithm 4. The intuition behind MWU-based aggregations is to reduce the weights of malicious parties using the distance between their malicious updates and the aggregated update. This is based on two assumptions: malicious updates lie farther away from the mean compared with the benign updates, and the number of malicious updates is smaller than that of the benign updates. Therefore, MWU-based aggregation schemes have the breaking point of $\lfloor \frac{n-1}{2} \rfloor$ malicious parties.

Algorithm 4 Multiplicative weights update: f_{MWU}

Input: $\mathcal{P} = (\theta_1^t, ..., \theta_n^t)$
Initialize parties' weight vector $\mathbf{w}^0 \leftarrow \mathbf{1}$ and $\theta_a^0 \leftarrow f_{\mathsf{AGG}}(\mathbf{w}^0, \mathcal{P})$ at $t = 0$
repeat
  $\mathbf{w}^{t+1} \leftarrow$ WeightUpdate ($\mathbf{w}^t$, θ_a^t, $\mathcal{P}$)
  $\theta_a^{t+1} \leftarrow f_{\mathsf{AGG}}(\mathbf{w}^{t+1}, \mathcal{P})$
  $t \leftarrow t + 1$
until Convergence criterion is satisfied
Output: final θ_a

There are different variants of MWU that use different functions for WeightUpdate and f_{AGG} in Algorithm 4. We detail two of them next.

A.4.1 MWU with mean aggregation

In MWU, if the weighted mean is used as the aggregation algorithm, f_{AGG} in Algorithm 4, it is called MwuAvg. Here, the weight of the i^{th} party is updated based on the distance between the weighted average, θ_a^t, of all the updates and θ_i; this is given by (4). The weights of all the parties are equal at the beginning.

$$w_i^{t+1} = w_i^t \exp(-||\theta_a^t - \theta_i||_p) \tag{4}$$

$$\theta_a^{t+1} = \frac{\sum_{i=1}^n w_i^{t+1}\theta_i}{\sum_{i=1}^n w_i^{t+1}} \tag{5}$$

A.4.2 MWU with optimization

Li et al. [25] propose a truth discovery framework CRH, to aggregate the responses in a crowdsourcing setting. In each epoch, the framework updates the weights of parties based on the solution of an optimization problem. Essentially, the weight update algorithm considers the distance of parties' updates from the aggregate of all the updates in each epoch. The weight update and aggregate computation are given by (6) and (7), respectively. For further details of the aggregation, please refer to [25].

$$w_i^{t+1} = -\log\left(\frac{||\theta^t - \theta_i||_p}{\sum_{i=1}^n ||\theta^t - \theta_i||_p}\right) \tag{6}$$

$$\theta^{t+1} = \sum_{i=1}^{n} w_i^{t+1}\theta_i \tag{7}$$

B Attacks on Aggregation Algorithms

In this section, we detail the poisoning and membership inference attacks used in our work to evaluate the robustness and privacy in federated learning. The poisoning attacks are of two types: *availability* and *targeted* attacks. The earlier attacks aim to jeopardize the overall accuracy of the final model/s [6, 49, 50], while the latter attacks aim to mis-classify only a specific set of samples of the attacker's choice at the test time [12, 11]. We focus on the poisoning availability attacks and below introduce such attacks from the literature, and also introduce a new poisoning attack targeting the MwuAvg and MwuOpt aggregations.

B.1 Label flip poisoning (Label flip)

We consider a type of data poisoning attacks [51, 52, 49], where the adversary flips the labels of her local training data in a particular fashion to poison it. We call this attack *label flip poisoning* attack. The label flipping strategy is performed consistently across all of the ϵn malicious parties that the adversary controls, i.e., all the malicious parties flip labels in the exact same way. Then, the ϵn malicious parties (also called as Byzantine workers in the literature) use this poisoned data to train their local models and then share corresponding updates with the central server.

B.2 Naive poisoning (PAF)

Our threat model considers an omniscient adversary who knows the distribution of benign updates, i.e., the mean and standard deviation of each dimension of benign updates. The adversary can estimate this distribution as she possesses data drawn from the distribution same as that of benign parties. Given this, the adversary crafts the malicious update θ_m to be arbitrarily far from the mean of the benign updates:

$$\theta_m = \frac{\sum_{i=1}^{(1-\epsilon)n}\theta_i}{(1-\epsilon)n} + \theta' \tag{8}$$

This malicious update, θ_m, is then shared by each malicious party with the central server. In (8), θ' is a vector of size $|\theta_i|$ with arbitrarily large (or small) coordinate values. This attack can jeopardize the weighted averaging, and interestingly, weighted median aggregations based federated learning.

B.3 Little is enough attack (LIE)

Baruch et al. [6] propose an attack called *little is enough* (LIE). The attack successfully circumvents state-of-the-art robust aggregation algorithms, including Bulyan and Krum. These aggregations are vulnerable to the attack, because they are tailored to an adversary that crafts a malicious update with at least one arbitrarily large dimension. However, in practice, a malicious update, θ^m, obtained by small perturbations in a large number of dimensions of a benign update suffice to affect model's convergence and also circumvent the defenses. Therefore, note that, *the root cause of the success of the LIE attack is also the high dimensionality of the updates.* The attack is described in Algorithm. 5. The s in the Algorithm. 5 is the number of non-corrupted parties, whose deviation from mean is more than malicious parties, needed in order to bypass the detection. Assume the distributions of different parameters from all parties can be expressed by a normal distribution, z will set the range in which the adversary updates can deviate from the mean without being detected. $\bar{\mu}, \bar{\sigma}$ are the mean and variance of the distribution of the parameters from benign updates.

B.4 Our poisoning attack (OFOM)

In this section, we propose an attack which targets aggregation schemes that perform weighted aggregation of data by assigning the weights based on the distance of the data points from an aggregate of the data. These aggregations are robust to all the above attacks, as we show in our evaluation. We discussed two such aggregation schemes: MWU with averaging [24] and MWU with optimization [25] in Section A.4. In any given epoch, the aforementioned aggregation schemes start with equal weights to all parties and update a party's weight based on the distance of the party's update from weighted average of all party updates. *The attack exploits the fact that, all parties are given equal weights to start with.* The OFOM attack craft two malicious updates: The first update, θ_1^m, is arbitrarily far away from the true mean, and is obtained by adding an arbitrarily large (or small) vector θ' to the mean of benign updates. The second malicious update, θ_2^m, is at the empirical mean of benign updates and θ_1^m. The malicious updates are formalized in (9).

$$\theta_1^m = \frac{\sum_{i=1}^{(1-\epsilon)n} \theta_i}{(1-\epsilon)n} + \theta', \quad \theta_2^m = \frac{\sum_{i=1}^{(1-\epsilon)n} \theta_i + \theta_1^m}{(1-\epsilon)n + 1} \tag{9}$$

This way, at the end of the first epoch of the MWU aggregation, the adversary manages to assign a weight close to 1 to the parties with update θ_2^m. In the case of MWUAvg and MWUOpt, all the benign parties are assigned negligible weights, which completely jeopardizes the accuracy of aggregation. To be effective, the adversary needs just two malicious parties who share the two malicious updates.

B.5 Membership inference attacks

Recent research has shown the susceptibility of the federated learning to active and passive inference attacks [7, 26]. In the passive case, the attacker, either the server or some of the parties, simply

Algorithm 5 Little is enough attack (LIE) [6]

1: **Input**: n, ϵ, mean and variance vectors of benign updates $\bar{\mu}$, $\bar{\sigma}$
2: Number of workers required for majority:

$$s = \lfloor \frac{n}{2} + 1 \rfloor - \epsilon n$$

3: Using z-table, set $z = \max_z \Big(\phi(z) < \frac{(1-\epsilon)n - s}{(1-\epsilon)n} \Big)$
4: **for** $j \in [d]$ **do**
5: $\quad \theta_m^j \leftarrow \bar{\mu}_j + z\bar{\sigma}_j$
6: **end for**
7: **Output**: malicious update θ_m

observes the updated model parameters to mount membership inference attacks. In the active case, however, the attacker tampers with the training of the victim model/s in order to infer membership of target data in *any of the benign party's data*. Specifically, the attacker shares malicious updates and forces the victim model/s to share more information about the members of their training data that are of the attacker's interest. This attack, called *gradient ascent* attack [26], exploits that the SGD optimization updates model parameters in the opposite direction of the gradient of the loss over the private training data. Let $\mathbf{x}$ be a record of attacker's interest and θ_a be the current global model. The attacker *crafts the malicious update θ^m by updating parameters of θ_a in the same direction of the gradient of the loss on* $\mathbf{x}$, i.e., performs gradient ascent as: $\theta^m = \theta_a + \gamma \frac{\partial L_{\mathbf{x}}}{\partial \theta_a}$. Such θ^m, when combined with the benign updates, increases the loss of the resulting global model, θ'_a, on $\mathbf{x}$. If $\mathbf{x}$ is in the training data of some party, SGD on θ'_a by this party will sharply reduce the loss of $\mathbf{x}$. On the other hand, if $\mathbf{x}$ is not in any party's training data, the loss of $\mathbf{x}$ will remain almost unchanged. Therefore, this attack increases the gap between the losses of θ_a on members and non-members and facilitates membership inference.

C Robust mean estimation

Cronus uses the robust mean estimation algorithm proposed by Diakonikolas et al. [16], which achieves the optimal error bound. The original algorithm is described in Algorithm 6. f_{Cronus} applies RobustFilter on the prediction of each public data. Recall that the intuition of this robust mean estimation [16] is that the empirical mean of the uncorrupted points should be concentrate nicely to the true mean μ of the distribution. Thus, if the empirical mean is $\hat{\mu}$ far from the true mean of the distribution, then along the direction $\hat{\mu} - \mu$, the outliers must be the source of the deviation and the variance is much larger than it should be. As a result, corrupted direction $\hat{\mu} - \mu$ can be detected as a large eigenvector of the empirical covariance. Hence, in the Algorithm 6 finds the direction v^*, which has the largest variance and projects the deviation of all the inputs from the empirical mean in this direction. Then filter out a randomized fraction of the data which are farthest from the mean, $\bar{Y}$, along this direction. Repeat the process until the variance is not large in every direction and then output the sample mean on the subsets. For further details of the robust mean estimation, please refer to [16, 17].

Algorithm 6 RobustFilter [Algorithm 3 [16]]

1: **Input**: S, ϵ, k=0
2: **while** True **do**:
3: Compute $\bar{Y}$,Σ, the mean and covariance matrix of S.
4: Find the eigenvector v^* with highest eigenvalue λ^* of Σ
5: **if** $\lambda^* \leq 9$ **then**
6: Let $\bar{Y}$ = $\bar{Y}$
7: break
8: **else**
9: Draw Z from the distribution on [0,1] with probability density function $2x$
10: Let T = $Z \max\{|v^* \cdot (Y - \bar{Y})| : Y \in S\}$.
11: Set $S = \{Y \in S : |v^* \cdot (Y - \bar{Y})| < T\}$
12: **end if**
13: **end while**
14: **Output:** $\bar{Y}$

The sample complexity of the Algorithm 6 is $\Theta(d/\epsilon \log d)$, where d is the dimensionality of the inputs. In the federated learning, d is the size of the model and in Cronus, d is the size of prediction. Table 4 shows the sample complexities for training different ML benchmark models with federated learning versus Cronus, using RobustFilter's algorithm RobustFilter. We note that Cronus significantly reduces sample complexity (by an order of 10^5), and therefore, unlike any existing federated learning, can achieve strong theoretical error guarantees even with a small number of parties in the network.

For the theoretical analysis, Algorithm 6 uses randomized filtering (step 9) and repeats until the stop condition is satisfied (step 5). The follow-up works from the same authors [17, 53] suggest a simpler algorithm to obtain a better performance in practice: (1) in each iteration, remove a deterministic fraction of the data instead of a random fraction. (2) repeat the filter for constant iterations in total. In

Table 4: Sample complexity, $\Theta((d/\epsilon)\log d)$, of Cronus aggregation, using [16], for parameters and predictions updates. For parameters, d is size of model; for predictions, d is the number of classes in the classification task. The ratio shows that Cronus learning can achieve the same error guarantee as in federated learning, but with a network which is 5 orders of magnitude smaller, for benchmark ML tasks.

Dataset	Sample complexity ratio of Federated learning over Cronus
SVHN	1.2×10^5
MNIST	3.3×10^5
Purchase	2.75×10^5
CIFAR10	10.4×10^5

Table 5: Evaluation of the conventional federated learning with various aggregation schemes with Cronus learning using the strong poisoning attacks described in Section 5. Robustness in Table 3 is measured as the ratio of the accuracy of the final model/s when the strongest attack is mounted and the accuracy in the benign setting; the strongest attack is determined empirically as the one that maximally reduces the accuracy of the corresponding federated learning aggregation.

Dataset		Federated learning with various aggregation algorithms						Cronus
		Mean	Median	MwuAvg	MwuOpt	Bulyan	Krum	
SVHN	Accuracy (Benign)	**95.9**	94.8	93.9	94.4	94.5	89.6	91.1
	Label flip	92.9	90.1	91.2	89.3	93.9	88.6	89.8
	LIE	14.8	14.5	91.6	92.0	15.5	16.2	91.5
	OFOM	0.9	94.5	0.9	0.7	94.4	89.0	91.0
	PAF	12.8	16.4	95.1	93.1	93.4	87.5	91.1
MNIST	Accuracy (Benign)	96.7	96.5	**97.2**	97.4	96.9	93.3	95.2
	Label flip	96.3	94.4	94.7	93.6	96.8	89.9	95.0
	LIE	95.1	93.1	95.6	96.7	94.1	94.3	95.9
	OFOM	22.1	97.3	25.3	36.0	97.1	94.4	96.1
	PAF	9.6	91.5	96.9	12.7	97.1	94.0	96.2
Purchase	Accuracy (Benign)	93.3	93.0	**93.6**	92.5	92.8	72.1	89.6
	Label flip	88.9	89.9	63.4	67.6	91.7	74.8	88.0
	LIE	2.5	69.3	92.2	85.6	81.8	49.6	89.2
	OFOM	1.4	92.8	1.8	1.1	92.6	74.5	89.4
	PAF	1.1	12.5	93.0	88.0	91.0	76.6	89.4
CIFAR10	Accuracy (Benign)	88.4	**89.1**	86.2	87.6	89.0	84.5	80.1
	Label flip	–	–	–	–	–	–	79.8
	LIE	18.9	61.2	86.0	84.3	75.6	18.0	78.0
	OFOM	12.9	89.5	14.2	12.8	89.1	85.0	78.5
	PAF	11.3	15.1	86.4	85.0	89.0	839	79.0

the evaluation, we filter out $\epsilon/2$ fraction of the inputs in each iteration (step 12) and repeat the filter process 2 times (step 7) and to obtain a good performance.

D Missing Experimental Details

D.1 Details of datasets and model architectures

We use four datasets in our evaluation, whose details follow.

SVHN. SVHN [36] dataset contains Google's street view images of house numbers. The images are 32x32, with 3 floating point numbers containing the RGB color information of each pixel. We use the extended SVHN dataset with 630,420 samples to train 32 party models each with 5,000 training samples; the public data size is 10,000. We use validation and test data of sizes 2,500 each. The reference data required for adversarial regularization is of the same size as that of training data for the cases of all the datasets.

MNIST. MNIST [37] dataset contains 28x28 images of handwritten digits and is composed of 60,000 training samples and 10,000 test samples. The dataset contains 10 classes each with 60,000

training and 1,000 test samples. We use validation and test data of sizes 1,000 each. We use 28 parties each with 2,000 training and reference samples, and public data size is 10,000.

Purchase. Purchase [54] dataset contains the shopping records of several thousand online customers, extracted during Kaggle's Acquire Valued Shopper challenge [54]. The dataset contains 197,324 data records with feature vectors of 600 dimensions and corresponding class label from one of total 100 classes. We use validation and test data of sizes 2,500 each. We use 16 parties each with 10,000 training and reference data, and public data size is 10,000.

CIFAR10. CIFAR10 [35] has 60,000 color (RGB) images (50,000 for training and 10,000 for testing), each of 32×32 pixels. The images are clustered into 10 classes based on the objects in the images and each class has 5,000 training and 1,000 test images. We use validation and test data of sizes 2,500 each. We use 16 parties each with 2,500 training data, and public data size is 10,000.

Model architectures. For SVHN, we use a neural network with three convolution layers and one fully connected layer, and Relu activations. For the MNIST dataset, we use a fully connected neural network with layer sizes {784, 256, 64, 10} and Relu activations. For the Purchase dataset, we use fully connected neural networks with layer sizes {600, 1024, 100} and Tanh activations. For CIFAR10 dataset, we use DenseNet architecture [55] with 100 layers and growth rate of 12. For the heterogeneity experiments with Purchase dataset, we use 5 fully connected networks with hidden layer sizes [{},{1024},{512, 256},{1024, 256}, {1024, 512, 256}]; here {} implies that the corresponding model has no hidden layers.

Training hyper-parameters. The initialization and collaboration phases of SVHN, MNIST, and Purchase trained models are of 50 epochs each. In both the phases, we train party models on their local training data using Adam optimizer at 0.0005 learning rate. Additionally, in collaboration phase, i.e., for epochs 50-100, we train party models on public data, $(X_p, \bar{Y})$, using SGD optimizer at a learning rate of 0.001. For CIFAR10, we train models for 200 epochs using SGD optimizer with 0.1 learning rate, 0.9 momentum, and 10^-4 weight decay in both the phases. Additionally, in collaboration phase, we train the models on public data using SGD optimizer at 0.01 learning rate, 0.99 momentum, and 10^{-6} weight decay.

For experiments of membership privacy assessment, we use state-of-the-art whitebox inference model proposed in [26], and use the gradients and outputs of its last layer, in addition to the blackbox access features including prediction of input and its cross-entropy loss. We train the inference model using Adam optimizer at a learning rate of 0.0001 for 100 epochs.

D.2 Missing Empirical Results

In this section, we provide the experimental details omitted in Section 5.

D.2.1 Robustness

Here, we give the complete robustness assessment of Cronus and FedAvg. In Table 3 of Section 5, for each dataset and each aggregation algorithm, we showed the accuracy of the attack that is strongest among all the attacks discussed in Section 5. We compute empirical robustness of aggregation algorithms using this strongest attack as described in Section 5. In Table 5, we give the complete evaluation of all the attacks on all of the aggregation algorithms and datasets we consider in this work. The 'Accuracy (Benign)' row of each dataset shows the results in the absence of adversary. The worst accuracy for a combination of aggregation algorithm and dataset is highlighted in the corresponding column; the corresponding strongest attack can be found from the label of the row of the highlighted cell. Observe that, label flip attack seems to have lower effect on mean aggregation than the other aggregations; this is because, unlike other aggregations, in case of mean, there is only single malicious client. Note that, MWUAvg and MWUOpt aggregations are robust against all the existing attacks in the literature, but are completely ineffective against the attack we introduced in Section B.4. Also, note that, Bulyan aggregation is empirically the most robust aggregation after Cronus, but it allows only 25 - 33% malicious clients compared to other aggregation algorithms such as Krum, in other words, Bulyan has a very low breaking point. The numbers of malicious parties used in each of our experiments are given in Table 2.

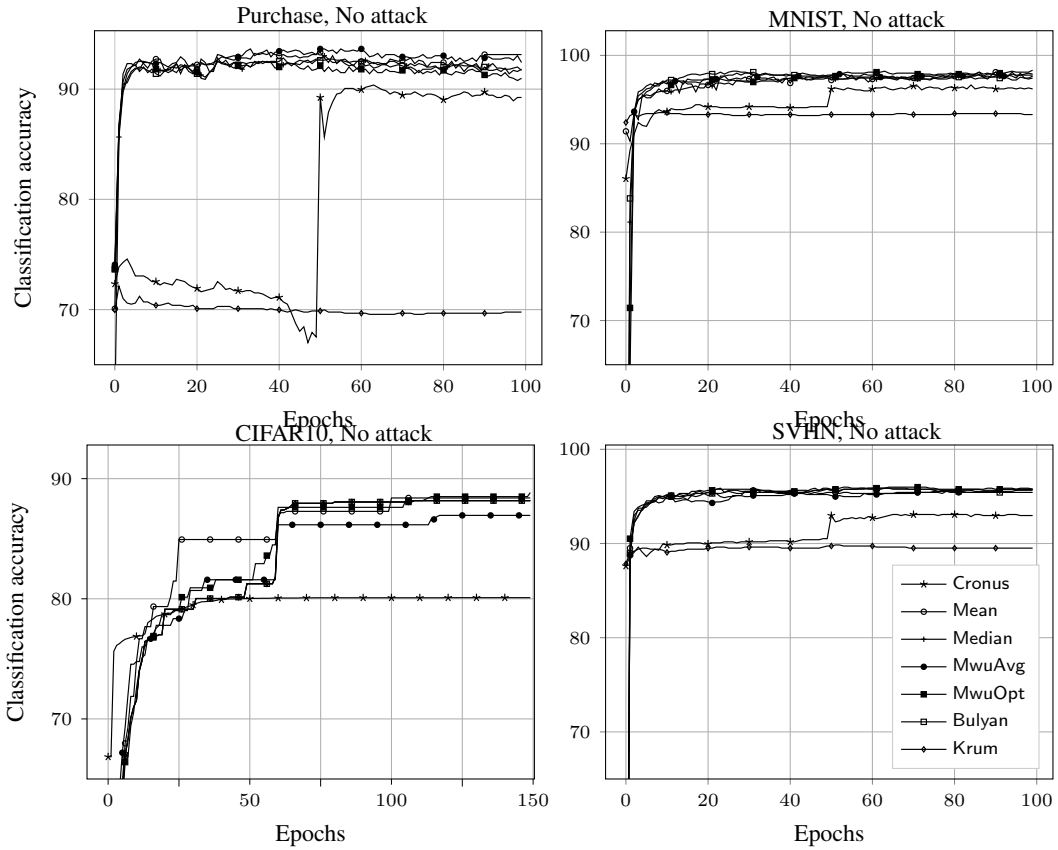

Figure 1: Convergence of Cronus and existing federated learning algorithms in **benign setting**. Cronus incurs only a slight degragation in accuracy compared to existing algorithms, while improves significantly over stand-alone training.

Convergence plots. Figure 1 shows the convergence of Cronus and various aggregation algorithms in federated learning for the benign setting. In Figure 2, However, all of the existing aggregation schemes in federated learning fail to converge under at least one poisoning attack. Cronus incurs only a slight reduction in accuracy at a significantly higher gain in robustness and privacy as shown in Sections 5 and 5.

D.2.2 Privacy

In this section, by measuring the privacy risk using state-of-the-art membership inference attack, we show that Cronus considerably reduces the privacy risk compared with federated learning. We also show the compatibility of Cronus with existing privacy-preserving mechanisms. We evaluate the risk of membership inference attacks on the participants' private training data during collaboration, and the effect of privacy preserving mechanisms [34, 56]. As described in the threat model in Section 2, we assume the central server and other participants to run passive and active membership inference attacks [26] on individual party updates and their aggregates. We use Purchase, SVHN, and CIFAR10 datasets for our evaluation when 4 parties collaborate.

Passive membership inference attacks

In the case of passive membership inference attacks, the server isolates the parties and mounts the attack separately on each of the collected updates, i.e., in case of FedAvg, attack is mounted on the parameter updates of each party and in case of Cronus, attack is mounted on the model obtained by training on the predictions shared by each party. We also evaluate the privacy risk when the attack is mounted on the aggregate of these updates.

The results are shown in Table 6. **The updates in FedAvg are highly susceptible to membership inference unlike the updates in Cronus.** For the Purchase dataset, attack accuracy against the individual and aggregated updates in FedAvg is 78.1% and 80.1%, respectively, whereas in Cronus

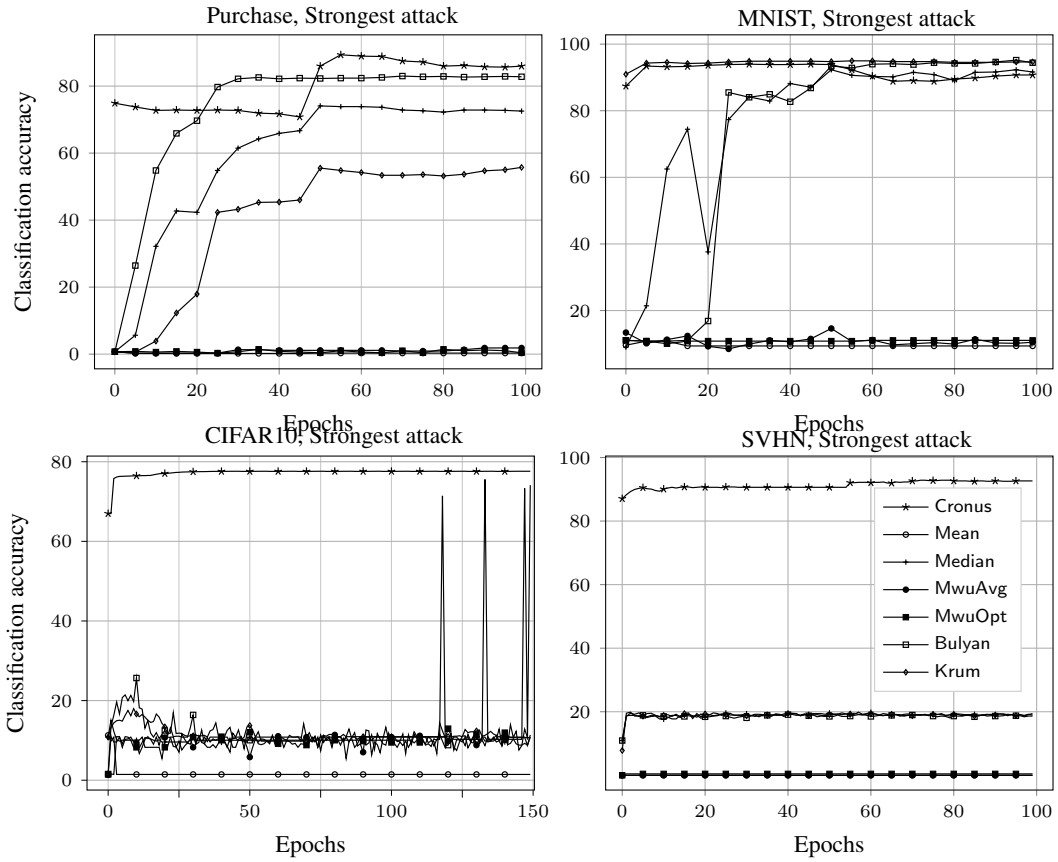

Figure 2: Convergence of Cronus and existing federated learning algorithms in **adversarial setting**. Accuracy of Cronus in adversarial setting is almost the same as in benign setting (shown in Figure 1) due to its high robustness. Except for CIFAR10, for which only collaboration phase is shown, both the Cronus training phases are shown in figure and the collaboration phase starts at epoch 50.

they are 51.7% and 51.9%. Unlike the prediction updates in Cronus, the high dimensional parameter updates in FedAvg encode a significantly higher amount of information about the party's local data. Furthermore, knowledge transfer acts as a strong regularization method and mitigates the risk of membership inference attacks [18, 57]. It's important to note that knowledge transfer through predictions, on a dataset other than the training data, makes the behavior of the student model more indistinguishable on its training versus unseen data. This happens as the distillation process does not carry the exceptionally distinguishable characteristics of the model on its training data, and results in smooth decision boundaries of the student model around the teacher's training data. We observe similar results for SVHN and CIFAR10 datasets.

We also evaluate Cronus and FedAvg, when models use adversarial regularization [56] during local training to improve membership privacy. We note that, **adversarial regularization improves membership privacy of both FedAvg and Cronus**, but the increase is smaller for Cronus due to its inherent resilience to membership inference. For Purchase dataset with FedAvg, the risk to individual and aggregated updates reduces by 9.9% and 12.7%, respectively, while for Purchase with Cronus, the risk to individual and aggregated updates is already very small, and it further reduces by 0.6% and 1.1%, respectively. Similarly for CIFAR10, privacy improvement in FedAvg is significantly more than in Cronus. However, the privacy improvement for SVHN is very small even in case of FedAvg, due to large gaps in train and test accuracies at stronger adversarial regularization.

Active membership inference attacks

In each epoch, the server manipulates the aggregate update that it broadcasts to parties by performing gradient ascent on the *aggregated update* for a set of target data [26]. In FedAvg, gradient ascent is performed directly on the aggregated parameters. In Cronus, for running such attack, the server needs to train a model on the aggregated predictions while performing gradient ascent on the target data,

Table 6: Accuracy of passive and active membership inference attacks with central server as adversary. We also evaluate effect of adversarial regularization (with parameter λ) used to preserve membership privacy. We use 4 parties and data per party is stated in the D.1.

Dataset	Federated learning algorithm	Attack on party update		Attack on aggregated update	
		Passive attack acc.	Active attack acc.	Passive attack acc.	Active attack acc.
Purchase (without privacy)	FedAvg	77.1	84.9	74.7	82.7
	Cronus	54.6	54.9	53.6	54.7
Purchase (with membership privacy, $\lambda = 3$)	FedAvg	70.8	77.3	69.9	77.0
	Cronus	54.1	51.5	53.7	54.6
SVHN (without privacy)	FedAvg	64.8	67.3	59.9	64.3
	Cronus	55.6	53.1	56.5	55.7
SVHN (with membership privacy, $\lambda = 0.5$)	FedAvg	64.9	67.0	60.0	64.2
	Cronus	54.1	56.9	55.6	55.0
CIFAR10 (without privacy)	FedAvg	79.9	80.5	76.8	77.1
	Cronus	57.0	57.8	55.5	56.7
CVIFAR10 (with membership privacy, $\lambda = 1$)	FedAvg	59.9	64.1	59.6	62.2
	Cronus	52.9	54.4	52.6	57.0

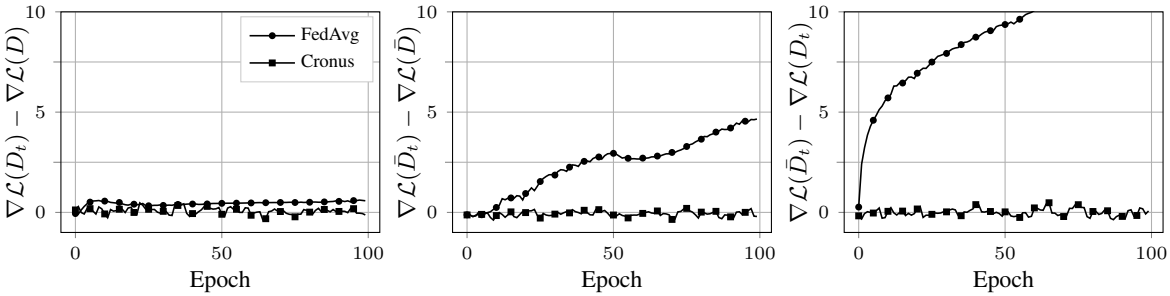

Figure 3: Difference in the gradient-norms, $\nabla\mathcal{L}(D)$, of the last layer of aggregated model on the target and non-target data (Purchase100 data). In the context of active membership inference attacks, D, D_t, $\bar{D}$, and $\bar{D}_t$ denote non-target members, target members, non-target non-members, and target non-members, respectively.

and then, shares predictions of this model with the parties; we ensure that such model has accuracy close to the accuracy of party models in given epoch.

Table 6 shows the results. **The active attacks significantly increase the privacy risk of the target data in case of FedAvg**: for Purchase dataset, the risk due to individual update increases by 7.8% (77.1% to 84.9%), while due to aggregated update increases by 8% (74.7% to 82.7%). But, **in case of Cronus, the active attacks are ineffective and the increase in risk is negligible**: 0.3% for individual update while 1.1% for aggregated update. In Figure 3, we show the effect of gradient ascent on the difference in gradient-norms of target and non-target data for aggregated model for the Purchase dataset. This directly correlates with the success of membership inference [26]. We observe that, for FedAvg on SVHN dataset, the active attacks increase the risk to individual and aggregate updates by 2.5% and 4.4%, respectively, but, the increased risk negligible for Cronus.

Differential privacy We compare the performance of Cronus and the conventional federated learning with user-level [58] (the server applies differentially private mechanism) or record-level [34] (each party applies differentially private mechanism) differential privacy. For both of these comparisons, we use SVHN dataset with 32 benign parties and the model architecture as described in Table 2, and use the moments accountant method (and the code) [34, 59] to bound the total privacy risk. Note that we train the *whole model* using differential privacy, as opposed to only training the last layer [34]. Our results show that robustness property of Cronus , as expected, is preserved with differential privacy.

User-level DP [58]. The user-level DP (UDP-FedAvg) algorithm proposed by McMahan et al. [58] cannot lead to training good accuracy models, when the number of parties in each epoch is small. Even for large privacy budgets, i.e., $\epsilon = 100$, the parties do not benefit from collaboration, and with the user-level DP, the global model in FedAvg achieves close to random-guess accuracy. We observed

Table 7: Accuracy and robustness of models for record level DP [34] with $\epsilon = 15.4$ on the SVHN dataset. The baseline stand-alone accuracy is 87%.

# of	Accuracy (Benign)		Worst accuracy		Robustness	
parties	FedAvg	Cronus	FedAvg	Cronus	FedAvg	Cronus
32	65.7	85.8	4.5	83.1	0.07	**0.97**
16	74.3	84.8	8.1	84.0	0.11	**0.99**
8	77.9	84.2	0.9	82.2	0.01	**0.98**
4	81.6	81.5	1.7	81.2	0.02	**0.98**

similar results for running user-level DP on Cronus with small number of participants. This result is expected as the sensitivity of the element-wise mean aggregation algorithm is inversely proportional to the number of parties, and a very large number of parties is required to reduce the noise, e.g., [58] uses 5000 parties in each epoch which is not realistic in cases where a few data holder (hospital) collaborate.

Record-level DP [34]. We empirically show that the conventional parameter aggregation in FedAvg is not suitable to provide the record-level DP, and is also susceptible to poisoning attacks. The results are shown in Table 7 for SVHN dataset. The accuracy of the models for federated learning or Cronus, with DP-SGD, cannot reach the accuracy of stand-alone training, which makes the collaboration useless. The results of the strongest poisoning attacks show that DP-SGD FedAvg has no robustness against the attacks.

D.2.3 Cronus with heterogeneous model architectures

Due to the use of predictions based updates, Cronus allows parties with heterogeneous model architectures to participate in collaboration. Below, we compare different aspects of the homogeneous and heterogeneous collaborations. We use Purchase data and 5 fully connected models, which we call A1, A2, A3, A4, and A5, with hidden layer sizes {}, {1024}, {512, 256}, {1024, 256}, and {1024, 512, 256} respectively. Note that, A1 models, called *bad models*, have lower capacity and accuracy than A2-5 models, which we call *good models*. We denote by $P_{j:k}$ the model architectures of parties $\in [j, k]$. We denote the entire collaboration in curly brackets, e.g., we denote the collaboration of 3 sets of 4 models, i.e. 12 models in total, each with either of A3, A3, or A4 models by $\{P_{1:4}$ = A2, $P_{5:8}$ = A3, $P_{9:12}$ = A4$\}$. In tables, accuracy of an architecture is the average accuracy of all the models with that architecture, e.g., in Table 9 accuracy of A2 is average of accuracies of all models with A2 architecture in the two collaborations.

First, we show that the heterogeneous collaboration between models of equivalent capacities does not reduce the accuracy of party models compared to its homogeneous counterparts. We consider four homogeneous collaborations each of 16 parties such that $\{P_{1:16}$ = A2$\}$, $\{P_{1:16}$ = A3$\}$, $\{P_{1:16}$ = A4$\}$, and $\{P_{1:16}$ = A5$\}$, and compare it with a heterogeneous collaboration: $\{P_{1:4}$ = A2, $P_{5:8}$ = A3, $P_{9:12}$ = A4, $P_{13:16}$ = A5$\}$. The results are shown in Table 8.

Next, we show that **the presence of a few bad models does not affect the accuracy of the good models in the heterogeneous collaboration, while significantly benefits the bad models**. Specifically, we show that accuracy of the collaboration of 12 good models, i.e., $\{P_{1:4}$ = A2, $P_{5:8}$ = A3, $P_{9:12}$ = A4$\}$ remains unaffected even if 4 bad models are added to it, i.e., $P_{13:16}$ = A2, as shown in Table 9.

Finally, we show that **heterogeneity allows for more knowledge sharing via collaboration and always improves the utility of collaborations**. We consider 4 homogeneous collaborations: $\{P_{1:4}$ = A1$\}$, $\{P_{1:4}$ = A2$\}$, $\{P_{1:4}$ = A3$\}$, and $\{P_{1:4}$ = A4$\}$ and compare them with a heterogeneous collaboration that includes all these 16 parties, i.e., $\{P_{1:4}$ = A1, $P_{5:8}$ = A2, $P_{9:12}$ = A3, $P_{13:16}$ = A4$\}$. Table 10 shows that including more participants clearly benefits all types of models, although the bad models benefit more than the good ones. For instance, A1 models improve by 8% from 70.1% to 78.1% due to heterogeneous collaboration, while A2, A3, and A4 models improve by 3.2%, 2.2%, and 1.4%, respectively.

Table 8: Comparison between heterogeneous and homogeneous collaborations in Cronus.

Homogeneous				Heterogeneous
$P_{1:16} \rightarrow$ A2	A3	A4	A5	{$P_{1:4}$ = A2, $P_{5:8}$ = A3 $P_{9:12}$ = A4, $P_{13:16}$ = A5}
89.6	89.3	88.4	88.6	89.3

Table 9: Effect of the presence of low accuracy bad models on the performance of higher accuracy good models. n is the number of collaborating parties.

Models	Heterogeneous {$P_{1:4}$ = A2, $P_{5:8}$ = A3, $P_{9:12}$ = A4}	Heterogeneous {$P_{1:4}$ = A2, $P_{5:8}$ = A3, $P_{9:12}$ = A4, $P_{13:16}$ = A1}
A1	-	78.1
A2	88.5	88.7
A3	88.6	88.1
A4	88.7	88.1

Table 10: More participation due to heterogeneity always improves the overall utility of the collaboration.

Models	Homogeneous 4 small collaborations $P_{1:4}$ = A1/A2/A3/A4	Heterogeneous {$P_{1:4}$ = A2, $P_{5:8}$ = A3, $P_{9:12}$ = A4, $P_{13:16}$ = A1}
A1	70.1	78.1
A2	85.5	88.7
A3	85.9	88.1
A4	86.7	88.1