Computationally Efficient Auto-Weighted Aggregation for Heterogeneous Federated Learning

Zahidur Talukder
University of Texas at Arlington
zahidurrahim.talukder@mavs.uta.edu

Mohammad A. Islam
University of Texas at Arlington
mislam@uta.edu

***Abstract*—Federated Learning (FL) offers a privacy-preserving massively distributed Machine Learning (ML) paradigm where many clients cooperatively work together towards training a shared machine learning model. FL, however, is susceptible to data heterogeneity problems as the FL clients have diverse data sources. Prior works employ auto-weighted model aggregation to mitigate the heterogeneity issue to minimize the impact of unfavorable model updates. However, existing approaches require extensive computation for statistical analysis of clients' model updates. To circumvent this, we propose, FedASL (Federated Learning with Auto-weighted Aggregation based on Standard Deviation of Training Loss) which uses only the local training loss of FL clients for auto-weighting the model aggregation. Our evaluation under three different datasets and various data corruption scenarios reveals that FedASL can effectively thwart data corruption from bad clients while causing as little as one-tenth of the computation cost of existing approaches.**

***Index Terms*—federated learning, model aggregation, data heterogeneity, computational efficiency**

I. Introduction

Motivation. Federated Learning (FL) has become a popular technique for privacy-preserving Machine Learning (ML). FL has been commercially deployed in several widely used applications such as Google Gboard's next-word prediction [1], Apple Siri's voice recognition [2], and WeBank's lending decisions [3]. Due to FL's privacy enhancement, it is of great interest in healthcare, where sensitive and private patient data is used to train ML systems [4].

In FL, a global model is cooperatively trained by participating users/clients who share their model updates based on local training with a central server that manages the global model [5]. The global model is updated iteratively, wherein each iteration, a subset of clients supply their model updates based on the latest global model. The clients' model updates are aggregated at the central server to generate the new global model for the next iteration. Fig. 1 illustrates the architecture of FL. Unlike centralized ML training, where all training data needs to be gathered in one place, in FL, the client's private data never leaves their devices. The clients only share the model updates from their local training with the central server. As a result, the client's privacy is greatly enhanced while at the same time, it can reap the benefit of a robustness ML model trained using many training samples (from all participating clients).

A critical step in FL is how the model updates are aggregated at the central server to construct the global model for the next round of training. Proper aggregation is crucial for the global model to converge and generate a stable model for reliable use (i.e., inference). The state-of-the-art model aggregation technique is called Federated Averaging (FedAVG), where weighted aggregation is used with each client's model update, receiving a weight proportional to the size of its training data set [5]. FedAVG has proven to be an effective yet straightforward aggregation technique for client data that exhibits iid (independent and identically distributed) characteristics. With iid client data, model aggregation can converge as clients' model updates do not deviate much from each other. However, such statistical coherence (i.e., iid client data) can seldom be ensured in practice. Different clients collect the training data for FL from various data sources. The data quality (e.g., noise) among clients may also vary significantly due to variations and aging of the hardware/sensors they use. The heterogeneous source of client data hinders both the accuracy of FL models and model convergence [6]. Another source of data heterogeneity is the presence of adversarial or compromised clients which intentionally send maliciously orchestrated model updates to cause model divergence and degrade overall accuracy [6], [7]. Despite the inherent weaknesses against heterogeneous data sources, FL offers unparalleled improvement of client privacy, making it a compelling approach for privacy-preserving ML in end-user-centric paradigms such as Internet-of-Things (IoT). Consequently, addressing the data heterogeneity problem has become a pressing issue toward widespread and generalized adoption of FL.

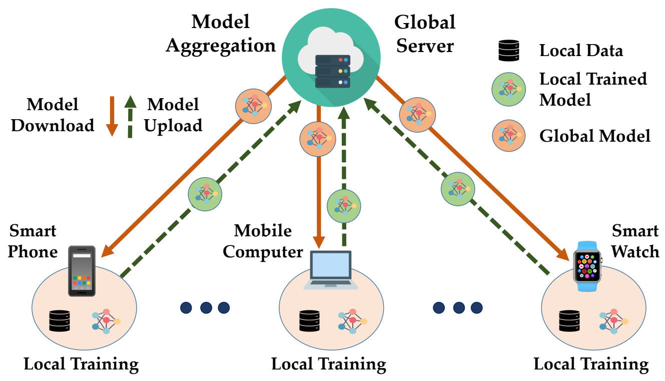


Fig. 1. FL architecture showing distributed learning using local training.

Limitations of existing approaches. The mainstream technique for addressing FL's heterogeneity problem is to dynamically tune the weights (i.e., auto-weighting) of the client updates in the central model aggregation [8]–[11]. The general approach in these techniques revolves around extracting aggregation weights using statistical analysis of all model updates. For instance, in the Trimmed Mean technique proposed in [9], values of each parameter of the ML model are individually updated by taking the average of client-supplied parameters after discarding the statistical outliers. Employing a similar fashion of model update, instead of using the average, [9] also proposes to use the median while [10], [11] use the geometric mean on the "clean" (i.e., without outliers) client parameters to construct the updated global model. Krum in [12], selects one of the user models as the representative of all model updates and uses it as the global model for the next FL iteration. Krum determines the representative model based on the model's Euclidean distance from other models. While these techniques have various degrees of effectiveness in thwarting the data heterogeneity issue, these are all computation heavy and do not scale efficiently as the number of participating users grows. Hence, these techniques are not suitable for resource-constrained central servers, for instance, in an edge-computing scenario where a lightweight edge-server acts as the central server to mediate FL model updates of users in geographical proximity. This computational burden can also be detrimental for hierarchical FL [13], where intermediate edge devices are responsible for aggregating user updates from their vicinity and coordinating with a central server. An alternative to the auto-weighting approach is active client selection [14], [15], where instead of handling the model updates after they are sent to the central server, the clients that send the model updates are carefully chosen to handle data heterogeneity. Active client selection can offer efficient communication over auto-weighting-based approaches. However, active client selection requires client profiling to determine the "good" clients, and hence, breaks FL's user anonymity. In this paper, we focus on the auto-weighting approach and improve its computation efficiency.

Our contribution. We circumvent the computation burden of the prior auto-weighting approach by avoiding extensive statistical analysis using the model parameters. Instead, in our novel approach, we only use the client's training loss to determine the weights of their model updates. More specifically, we ask each participating client in an update round to report their training loss along with their model updates. We take all clients' training losses and employ an auto-weighting based on how far a client's training loss is from the median loss. We set a "good region" around the median loss and assign equal weights to all clients' updates whose losses fall within the good region. Clients outside the good region receive weights inversely proportional to their distance from the median loss. More importantly, however, we determine the good region and the weight distribution among the clients inside and outside the good region based on the standard deviation of the clients' training loss. We call our approach FedASL (**Fed**erated Learning with **A**uto-weighted Aggregation based on **S**tandard Deviation of Training **L**oss).

FedASL saves on computation as it only needs to calculate the median and standard deviation of the client's local training loss. Also, our standard deviation-based weights change dynamically as the FL model update goes through update iterations. Moreover, FedASL acts as a filter to downplay the updates from clients with unfavorable data in the model aggregation. Hence, as FL progresses, the performance of our statistical approach improves, i.e., FedASL increasingly prioritizes the good updates over the bad ones.

We extensively evaluate FedASL with three prominently used data sets for FL - `CIFAR10`, `MNIST`, and `FEMNIST` using several different ways to introduce data heterogeneity. We compare FedASL with FedAVG as well as other state-of-the-art auto-weighting algorithms such as Trimmed Mean and Median [9]. We show that FedASL offers better or similar model accuracy compared to existing techniques while avoiding the heavy computation for auto-weighing with as little as one-tenth of their computation cost.

II. Preliminaries

A. Federated Learning

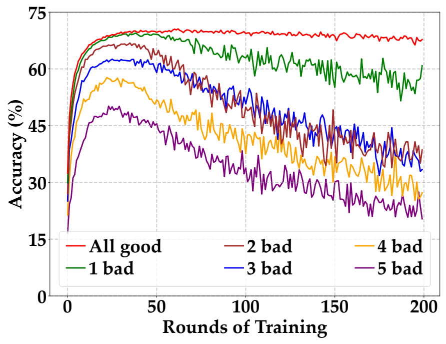
FedAVG is the state of the art of FL technique where local model updates are merged in a global server and sent back again to the clients for further training [5]. The model is updated synchronously in rounds of communication between the server and clients. We use neural networks as an example to discuss FedAVG. A neural network can be represented as a function $f(x, w) = y$ that maps input x to an output y, where $w \in \mathbb{R}^n$ is the model weight parameters of f. An observed pair $\langle x, y\rangle$ represents a training sample and a training set with m samples is a collection $D = \{\langle x_i, y_i\rangle\ i = 1, \cdots, m\}$ for a particular client. Suppose there are N devices and i_{th} device has a local training dataset D_i. Here, the aim of the client devices is to collaboratively learn the model parameter and minimize their defined loss. Specifically, the model weight parameter w is obtained by solving the optimization problem: $min_w \sum_{i=1}^{N} \mathcal{F}(w, D_i)$, where $\mathcal{F}(w, D_i)$ is the objective function of i_{th} device. The objective function may be different for different classifiers such as logistic regression and deep neural network. A popular choice for the loss function is squared L2 norm, i.e., $l_f(w, x_i, y_i) = (y_i - f(x_i, w))^2$.

Considering a certain round of training where the server selects K clients among the N clients at random, where $K \leq N$. The local dataset of the k_{th} client is D_k. Total data points of the selected client is $n = \sum_{k=1}^{K} |D_k|$. The objective function becomes

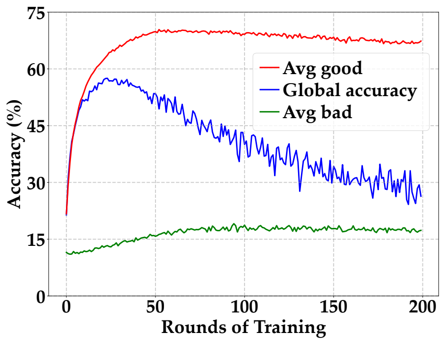
$$l(w) = \sum_{k=1}^{K} \frac{n_k}{n} \mathcal{F}_k(w, D_k) \quad \text{with}$$

$$\mathcal{F}(w, D_k) = \frac{1}{n_k} \sum_{i \in D_k} l_i(w, x_i, y_i) \quad (1)$$

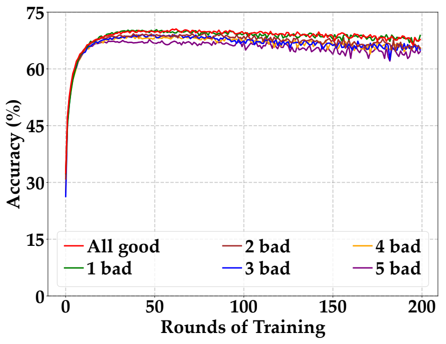
Minibatch stochastic gradient descent (SGD) can be used to solve Eqn. (1). In the FL setup, the server sends the global



(a) FedAVG



(b) Average of good and bad clients



(c) Zero weighting

Fig. 2. (a) Accuracy of the global model in FedAVG with different number of unreliable clients with bad data. (b) Average accuracy of the good and bad clients, the global accuracy degrades due to bad clients. (c) Impact of discarding bad client by giving no weights on the global model using the same data.

model to the selected clients. At round t, the client used the global model w_t as their initialization for local training. Now by using SGD, the local parameter is updated as follows-

$$w_t^k \leftarrow w_t - \eta \nabla \mathcal{F}(w_t, D_k) \quad \text{for all K clients} \tag{2}$$

where η is the learning rate and $\nabla \mathcal{F}(w_t)$ is the gradient of the loss function with respect to weight parameters. Now the global model for the round $t+1$ is updated by taking the weighted average of the local updates that are received from the local clients as the global model-

$$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_t^k \tag{3}$$

After updating the global model parameter, the global server selects a new random set of clients for further training with their local data. Hence, each individual client is responsible for training the model with their local data and returning the trained model to the server. Data privacy is greatly enhanced in FL since raw user data never leaves the user devices. Moreover, clients do not require user identity and hence can send their updates anonymously without any meta-data.

B. *Heterogeneity in FL*

The data heterogeneity is prevalent in FL as different clients collect the training data from different data sources using different hardware/sensors. In case of IoT and edge devices, the data collecting sensors are different both in quality and age and are vulnerable to cyber-security attacks [16]. It is observed that the quality of sensors degrades over time [17]. In such cases, the data generated from those devices may have corruption both in features and labels. In crowd-sourcing scenarios, on the other hand, the data can be noisy because of bad workers or biasing [18]. Moreover, due to the decentralized nature of FL, the data distribution of clients can also non-iid [19]. In worst-case scenarios, users can even be malicious or corrupted. So data from all edge devices are not of the same quality. Hence, if we blindly include these clients in FL with the same weights as the good/reliable clients, the global model performs poorly. We identify that the degradation of the global model is due to different data distribution of the corrupted client from good clients, causing model divergence. FedAVG algorithm takes the weighted average of the clients' model updates to construct the global model, and therefore, fails to remove the effect of bad and noisy clients on the system.

C. *Motivating Example*

As a foundation of our approach, we hypothesize that participation from unreliable clients with bad data quality is harmful to the overall accuracy and convergence in the central model aggregation. To see the effect of data heterogeneity in federated setup, we make a small experiment with only 10 clients.

Impact of client heterogeneity. To support our hypothesis, we run experiments on the `CIFAR10` data set with 10 clients. We vary the number of clients with corrupted data and check the accuracy of the global model after each communication round. We consider all data of an unreliable client are mislabeled. For better illustration, here we consider clients with 100% data corruption to amplify the impact of clients' data quality on the global model's performance. The model is trained using FedAVG's aggregation techniques. Fig. 2(a) shows the impact of bad clients on global accuracy. As the number of bad clients increases, accuracy and convergence greatly suffer. Since FedAVG does not have any mechanism to control the effect of bad clients in the system, there is no way to mitigate the effects of bad clients.

Solution approach. To identify the client behavior during training, we look into the accuracy of individual clients. We find that the accuracies of all good clients are close to one another and the accuracies of all the bad clients are behaving differently. Fig 2(b) shows that bad clients suffer from low model accuracy due to their bad data quality. The degraded performance of the bad clients also hampers global accuracy.

To support our hypothesis that the bad clients are responsible for waning global accuracy, we next test the same model by giving "zero" weights to the bad clients. In Fig 2(c), we show the global accuracy where the bad clients are essentially

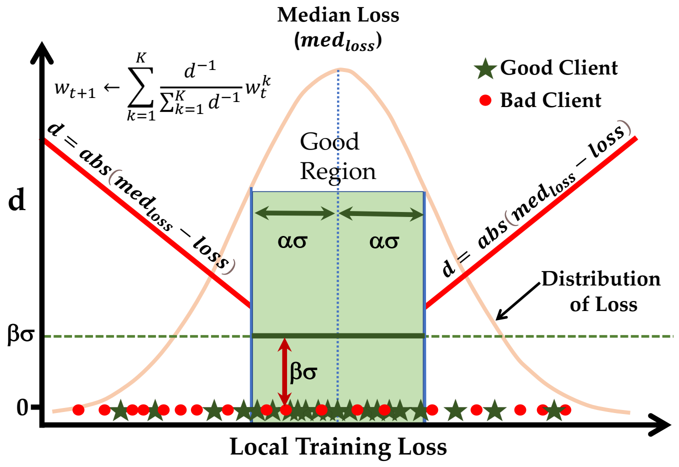

Median Loss
(med_{loss})
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{d^{-1}}{\sum_{k=1}^{K} d^{-1}} w_t^k$
Good Client
Bad Client
Good Region
d
$d = abs(med_{loss} - loss)$
$d = abs(med_{loss} - loss)$
ασ
ασ
Distribution of Loss
βσ
βσ
0
Local Training Loss

Fig. 3. FedASL architecture. Here the good region is $2\alpha\sigma$ and any clients which reside within this zone gets the value $\beta\sigma$. Clients inside the good region will always contribute more in the global model.

discarded from central model aggregation. Here, we see that the global accuracy and convergence are not adversely affected if we can identify and discard bad clients from model updates.

Challenges. Our motivating examples demonstrate that discarding clients with bad data is an effective approach to handling data heterogeneity problems. However, to employ this approach in practice, we need to identify the bad clients. Towards that, in the next section, we propose a novel and computationally efficient approach where statistical analysis of the user local update loss is utilized to separate bad clients. We develop a auto-weighted aggregation called FedASL where clients are penalized for their statistical abnormality (i.e., bad clients) by assigning lower weights during model aggregation.

III. FedASL

In this section, we present the details of FedASL (**Fed**erated Learning with **A**uto-weighted Aggregation based on **S**tandard Deviation of Training **L**oss).

Design principles. To enable a lightweight analysis, instead of entire models with many parameters, we use only one parameter per client - the client's local training loss. In each update round, we use the participating clients' losses to determine the quality of each client's model updates based on how far their loss is from the median loss. The clients' updates get weights inversely proportional to their distance from the median loss. However, directly using this approach is too restrictive as it penalizes all but one client with the median loss. In practice, there are many good client updates which should be treated equally. To implement this, we set a "good region" around the median loss and consider all clients with loss in the good region as good client. We determine the good region based on standard deviation of the model losses. For instance, we can set the good region around one standard deviation from the median loss. We also allow FedASL to tune the relative weights of clients inside and outside the good region. We use two tunable parameters, α and β, to control the size of the good region and the weights of the clients inside the good region.

Algorithm 1 FedASL Algorithm

```
Server Side:
Initialize($w_0, \alpha, \beta$)
for each round $t = 0$ to $T$ do
    Select a subset of $S_t$ from $N$ clients at random
    Broadcast the Global Model $w_t$ to the selected client $K$
    for each Client $k \in K$ do
        $w_t^k$, $L_t^k$ ← Client Update ($w_t$)
    end for
    Update $w_{t+1}$ according to Equation 9
end for
Client Side: // Run on selected client k on a training round
Client Update ($w_t$)
for each local epochs $i$ from 1 to $E$ do
    $w_t^k \leftarrow w_t - \eta \nabla \mathcal{F}(w_t, D_k)$
    $L_t^k \leftarrow$ training loss
end for
return $w_t^k$ and $L_t^k$ // sent to server
```

Setting the good region. As described before, the good region is set based on the median and standard deviation of the clients' local training loss. We first define the median of the loss as

$$med_{loss} = \begin{cases} \mathcal{L}[\frac{K}{2}], & \text{if K is even.} \\ \frac{1}{2}\left(\mathcal{L}[\frac{K-1}{2}] + \mathcal{L}[\frac{K+1}{2}]\right), & \text{if K is odd.} \end{cases} \quad (4)$$

where $\mathcal{L} = \{L_1, L_2, \cdots, L_K\}$ is set of training losses reported by the K clients sorted in ascending order. The standard deviation σ is defined as

$$\sigma^2 = \frac{1}{K}\sum_{k=1}^{K}\left(L_k - \frac{1}{K}\sum_{k=1}^{K} L_k\right) \quad (5)$$

Using the median and standard deviation from Eqns. (4) and (5), respectively, we define the set of loss values in the good region, $\mathcal{L}_{good}$, as follows

$$\mathcal{L}_{good} = \{L_k : (med_{loss} - \alpha \cdot \sigma) \leq L_k \leq (med_{loss} + \alpha \cdot \sigma)\} \quad (6)$$

Here, the tunable parameter α determines how restrictive FedASL is in determining which updates to consider as good updates. A small value of α will make FedASL more selective and vice versa. With increasing value of α more and more client updates are included in the good region. Sufficiently large α renders FedASL to behave as FedAVG which considers all client updates as good. Also note that, since in every round a new set of clients participate in FL update, our good region changes dynamically in every FL round.

Setting the client weights. We set the weights of each client's based on the value of d which is defined as follows

$$d_k = \begin{cases} \beta \cdot \sigma, & \text{where } L_k \in \mathcal{L}_{good} \\ abs(med_{loss} - L_k), & \text{where } L_k \notin \mathcal{L}_{good} \end{cases} \quad (7)$$

Here, $0 < \beta \leq \alpha$ is another tunable parameter that controls the relative weight distribution among the clients inside and outside the good region. The weight parameter of each client's model update, A_k, is set as

$$A_k = \frac{d_k^{-1}}{\sum_{k=1}^{K} d_k^{-1}} \tag{8}$$

Note here that, $\sum_{k=1}^{K} A_k = 1$ which ensures overall weights remain the same across updates.

A smaller value of β in Eqn. (7) will increase the weights of clients inside the good region resulting in FedASL heavily discriminating clients outside the good region. The value of β is restricted to go above α to avoid clients inside the good region receiving lower weights than clients outside. With the auto-weighting of each client, the FL model update can be written as

$$w_{t+1} \leftarrow \sum_{k=1}^{K} A_k \cdot w_t^k \tag{9}$$

Fig. 3 illustrates the good region and value of d for FedASL while Algorithm 1 formally describes FedASL.

IV. Evaluation

We evaluate our FedASL with a basic FL setup of one server and N clients settings. In every communication round, a subset of clients is randomly selected. We evaluate our model with the base FedAVG, Trimmed Mean and Median algorithms in various data corruption scenarios with three different datasets.

A. Evaluation Settings

1) Dataset: We adopt three popular datasets MNIST, CIFAR10 and FEMNIST, which are commonly used in literature [6], [20].

MNIST: The MNIST dataset is a popular handwriting dataset that consists of 70,000 grayscale images of size 28×28 pixels each. The dataset is divided into 60,000 training samples and 10,000 test samples. The images are classified into 10 classes from 0 to 9. We uniformly divide the training dataset among 100 clients, each getting 600 samples. We use the original the test set for evaluating the model performance over time.

CIFAR10: The CIFAR10 is another popular dataset consisting of 60,000 colored images of size 32×32 pixels. The dataset is divided into 50,000 training images and 10,000 images. These images are grouped into 10 separate classes. We divide the dataset into 100 clients where each client gets 500 samples. We use the original test set for evaluating the model performance over time.

FEMNIST: The FEMNIST dataset is taken from Tensorflow federated which is derived from the LEAF [21] dataset. Here, the dataset is divided into 3,383 true users. There are 341,873 train examples and 40,832 test examples of 28×28 pixels gray images. The test set has at least one sample from each user. This is a non-iid and heterogeneous dataset where each user represents a different client.

TABLE I
Dataset Description and Model Used

Dataset	Training	Test	#Client	Distribution	Model
MNIST	60,000	10,000	100	IID	LR
CIFAR10	50,000	10,000	100	IID	CNN
FEMNIST	341,873	40,832	3383	Non-IID	LR

2) Model parameters: We focus on an edge setup where our clients are IoT devices. Hence, we choose simpler and lightweight models as IoT devices have limited power and computational capacity. The model parameters used for our model training are discussed below.

MNIST: For the MNIST dataset, we use simple logistic regression classifier with Tensorflow Keras sequential model. The input feature is flattened before training and we take the one hot encoding for the label. We use the 'softmax' activation function and L1=0.01 and L2=0.01 kernel regularizer, 'adam' optimizer and 'categorical-crossentropy' as loss function.

CIFAR10: For the CIFAR10 dataset, we use only 1 block VGG [22] network. The architecture consists of 3×3 convolutional layer (with 32 convolutional filters and 'relu' activation function), followed by 2×2 maxpooling, a dropout layer (0.2); followed by a dense layer of size 128, 'relu' activation function, dropout layer (0.5), and finally 'softmax' activation function is used for finding 10 desired outputs. The model remains simple to work around in Edge IoT devices.

FEMNIST: For the FEMNIST dataset, we use simple logistic regression classifier with Tensorflow Keras sequential model. We use 'relu' activation function and a dense layer of size 128, 'adam' optimizer, and 'Sparse-categorical-crossentropy' as loss function. The model size remains small based on our target the IoT devices.

3) Evaluation scenarios: We consider that due to the quality and aging of the sensors, there could be 3 different ways the user data can be corrupted as described below.

Label Shuffling: In the label shuffling case, the label interpretation of the corrupted sensor is wrong. So the sensor labels the data randomly. In our experiment, we also vary the percentage of clients whose label is randomly shuffled.

Label Flipping: In the label flipping case, the label from a client is flipped to a random value i.e. all the labels in a corrupted client is the same. In our experiment, we consider a fraction of the sensors always give a fixed random label output.

Noisy Data: In the noisy data scenario, labels are interpreted properly but the feature space is considered noisy. To mimic this type of data corruption, we add Gaussian noise to features. Before that, we normalize the input to the interval [0, 1]. In this case, for the selected client we add Gaussian noise $x = x + \epsilon$, where $\epsilon \sim N(0, 0.7)$. Then we normalize the resulting value again to the interval of [0, 1].

4) Benchmark algorithms: To evaluate FedASL's performance, we compare it with the following state of the art techniques:

- FedAVG **[5]:** This is the standard federated averaging techniques where client weights are determined based on their dataset size.

TABLE II
ACCURACY OF ALL FL ALGORITHMS UNDER DIFFERENT LEVELS OF DATA CORRUPTION. IN EACH ROUND, 30% OF CLIENTS PARTICIPATED IN CIFAR10 AND MNIST, AND 10% IN FEMNIST.

	Data Type	Clean	Shuffling			Flipping			Noisy		
	Bad Client Per.	0%	10%	30%	40%	10%	30%	40%	10%	30%	40%
CIFAR10	FedAVG	**63.55**	59.83	53.53	44.8	60.19	52.5	49.31	62.16	58.73	56.14
	Median	59.1	57.89	51.16	46.92	57.57	53.77	52.13	57.57	52.52	49.97
	Trimmed Mean	62.51	59.65	54.02	46.87	60.39	52.96	47.57	59.95	54.57	53.06
	FedASL	63.03	**63.32**	**61.38**	**60.69**	**62.77**	**61.52**	**60.46**	**63.49**	**62.09**	**58.48**
MNIST	FedAVG	**82.5**	82.02	76.05	71.64	81.77	80.15	79.5	82.16	80.63	79.9
	Median	82.4	82.15	80.89	78.95	82.28	81.9	80.69	82.07	80.44	79.1
	Trimmed Mean	82.42	82.16	78.52	72.27	82.24	80.8	79.56	82.1	80.46	79.38
	FedASL	82.43	**82.43**	**82.26**	**81.68**	**82.51**	**82.36**	**81.71**	**82.56**	**82.25**	**82.2**
FEMNIST	FedAVG	**78.25**	**77.58**	74.31	69.62	**77.43**	74.41	72.31	77.57	75.69	74.75
	Median	76.49	75.89	73.72	71.7	76.47	75.85	**76.01**	76.48	75.63	74.92
	Trimmed Mean	77.63	77.18	74.75	71.56	71.85	**75.88**	73.63	**77.68**	75.63	75.1
	FedASL	77.62	77.54	**76.73**	**76.12**	77.06	74.64	71.33	77.37	**77.09**	**76.47**

• Median **[9]:** This is one of the Byzantine robust aggregation rule that aggregates each model parameter independently. Specifically, for each i^{th} model parameter, the global server sorts the i^{th} parameters of the K selected clients i.e. $w_{1i}, w_{2i}, w_{3i}...w_{Ki}$, where w_{ki} is the i^{th} parameter of the k^{th} local clients. Then the server takes the median of the sorted parameters as the i^{th} model parameter for the global model.

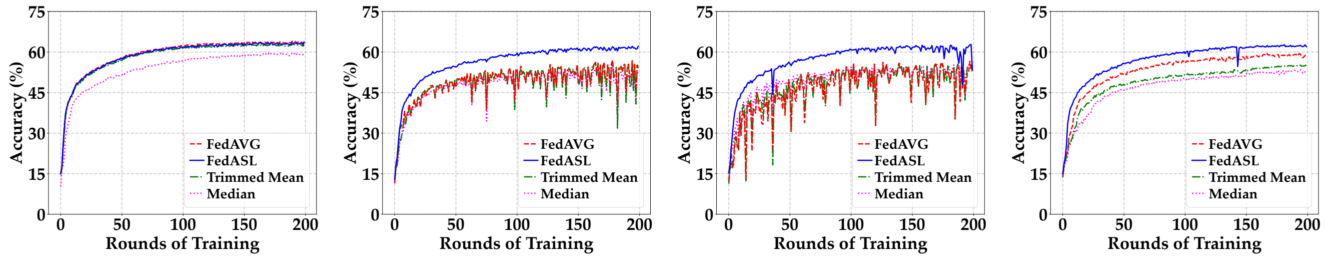
• Trimmed Mean **[9]:** This is another Byzantine robust aggregation rule that also aggregates each model parameter independently. Specifically, for each i^{th} model parameter, the global server sorts the i^{th} parameters of the K selected clients i.e. $w_{1i}, w_{2i}, w_{3i}...w_{Ki}$, where w_{ki} is the i^{th} parameter of the k^{th} local clients. Then the server removes the largest and smallest β of them and finally, takes the mean of the remaining $K - 2\beta$ as the i^{th} parameter of the global server.

B. Evaluation Results

1) Comparison with benchmarks: Table II shows the accuracy of different algorithms on the three different datasets for different cases with different percentages of bad clients. In each round of training, 30% of clients are randomly taken for CIFAR10 and MNIST, and 10% of clients are randomly taken for FEMNIST. As we can see in the table, our FedASL algorithm is robust under different data corruption scenarios and different corruption levels. It achieves the highest test accuracy in most cases. Especially, if we look at the cases for CIFAR10, the accuracy of the existing techniques decreases a lot with a little introduction of corrupted clients. As we can see, for the shuffling cases the test accuracy of FedAVG is only 44.8%, Median is 46.92% and Trimmed Mean is 46.87% for 40% corrupted clients, where our FedASL get 60.69% test accuracy which is very close to the accuracy in a clean environment. We see the same type of test accuracy in flipping cases also, for FedAVG is only 49.31%, Median is 52.13%, and Trimmed Mean is 47.57% for 40% corrupted clients, where our FedASL gets 60.46% test accuracy which is very close to the accuracy of the clean environment. In case of the noisy clients, we also find FedASL can achieve higher accuracy compared to the state-of-the-art techniques in all corruption levels. Even for the clean environment, our model does not degrade the model accuracy which indicates that we can use FedASL even when there are no bad clients.

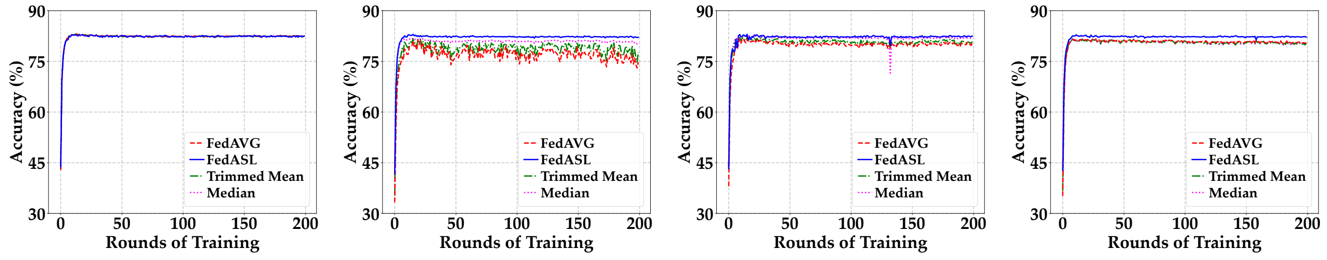
We see the same result for MNIST dataset as well. As we have used a simple logistic regression model for the MNIST dataset, the difference in test accuracy of different algorithms is less. Even though, in all the corrupted cases our FedASL algorithm outperforms the state-of-the-art techniques. As we can see for the shuffling cases the test accuracy of FedAVG is only 71.64%, Median is 78.95% and Trimmed Mean is 72.27% for 40% corrupted clients, where our FedASL gets 81.68% test accuracy which is very close to the accuracy of the clean environment. For the MNIST flipping cases also, our algorithm outperforms all the existing methods. In the noisy data cases, FedASL performs better compared to the other methods. Even for the clean environment, our method gives almost the best result which also proves that in the normal logistic regression model also, our model can be used in all test cases.

For the FEMNIST dataset, the data is non-iid and the number of data points among the client is also different. Nonetheless, in most cases, our model outperforms the state-of-the-art techniques. We also use the logistic regression for the FEMNIST dataset and for the shuffling cases the test accuracy of FedAVG is only 69.62%, Median is 71.7% and Trimmed Mean is 71.56% for 40% corrupted clients, where our FedASL gets 76.12% test accuracy which is very close to the accuracy of the clean environment. But for the FEMNIST flipping cases, we found that Trimmed Mean outperforms other for 30% corrupted clients and Median outperforms in 40% corrupted clients scenarios. But for the noisy cases, we can see with the increase in corrupted client percentage, our FedASL algorithm outperforms the existing state-of-the-art. Also for the clean environment, our model achieves quite a similar accuracy to the best method which implies, it's better to use FedASL even without the presence of any bad clients. One point to be noted is that mislabeling which we refer to in our paper as shuffling is the most expected type of data corruption compared to the flipping or same label for all the data in the same clients. Since our model can attain higher accuracy in shuffling and noisy scenarios, FedASL is the best



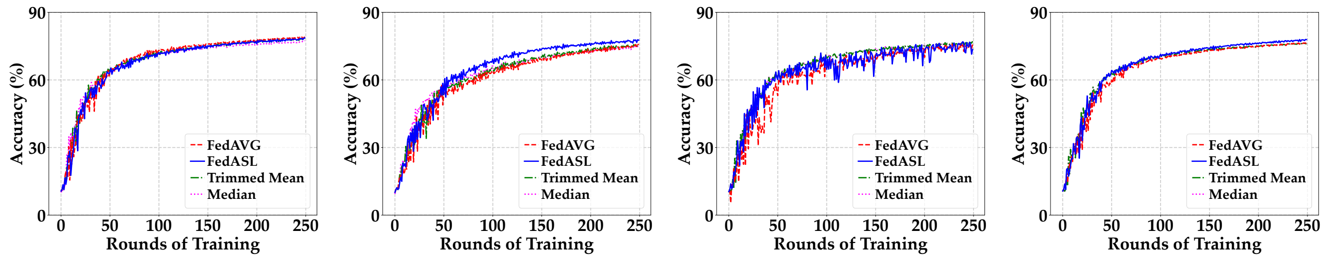
(a) Clean, $\alpha = 1$, $\beta = 0.2$ (b) Shuffling, $\alpha = 0.9$, $\beta = 0.2$ (c) Flipping, $\alpha = 0.9$, $\beta = 0.2$ (d) Noisy, $\alpha = 0.9$, $\beta = 0.2$

Fig. 4. Performance of different algorithms for `CIFAR10` data corruption for 30% bad clients. In each round, 30% clients are selected at random.



(a) Clean, $\alpha = 0.2$, $\beta = 0.05$ (b) Shuffling, $\alpha = 1$, $\beta = 0.2$ (c) Flipping, $\alpha = 0.9$, $\beta = 0.2$ (d) Noisy, $\alpha = 0.9$, $\beta = 0.2$

Fig. 5. Performance of different algorithms for `MNIST` data corruption for 30% bad clients. In each round, 30% clients are selected at random.



(a) Clean, $\alpha = 0.5$, $\beta = 0.2$ (b) Shuffling, $\alpha = 0.7$, $\beta = 0.2$ (c) Flipping, $\alpha = 0.1$, $\beta = 0.01$ (d) Noisy, $\alpha = 0.7$, $\beta = 0.2$

Fig. 6. Performance of different algortithms for `FEMNIST` data corruption for 30% bad clients. In each round, 10% clients are selected at random.

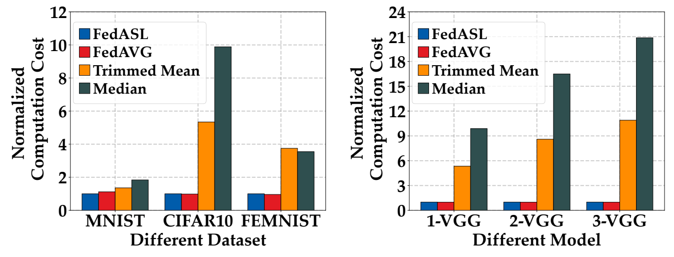
choice in data corruption scenarios even in non-iid cases.

2) Convergence analysis: We compare the convergence of our model with the existing state-of-the-art techniques in Figs. 4, 5, and 6, where we see the attaining accuracy vs the rounds of training needed for attaining that accuracy. The results are shown for 30% corrupted clients. As we can see in Fig. 4(a), for `CIFAR10` in the clean environment, all the algorithms converge smoothly even though Median based algorithm attains lower accuracy. For the `CIFAR10` shuffling and flipping cases, FedASL converges faster with high accuracy compared to the other techniques that struggle in convergence. And for the `CIFAR10` noisy cases also our model attain higher accuracy with smooth convergence.

For the `MNIST` dataset, all the models converge smoothly in a clean, flipping, and noisy environment. But for the shuffling cases, our model converges more smoothly than others and attains a higher test accuracy.

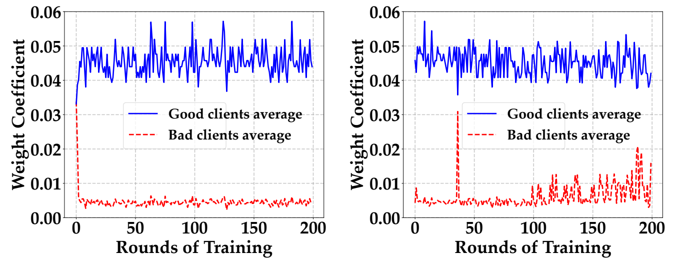
For the `FEMNIST` dataset, all the model converges smoothly for the clean and noisy cases. But for the shuffling and flipping cases, all the methods struggle a bit to converge. But in flipping cases FedASL struggles more as in flipping cases bad clients trends to be more in the α zone and removing them by statistics becomes harder. And with proper tuning α and β values, we expect to attain a better result.

3) Computation cost analysis: To compute the computation cost of the different algorithms, we see the server-side computation time. We first run the algorithms in three models for the three datasets. As mentioned earlier, we use a very small model for `MNIST` and `FEMNIST` and those are the logistic regression model. We find that the computation time in server side for the FedAVG and FedASL is almost similar as FedAVG needs to calculate the weighted sum of the model as the global model on the server side and our model FedASL needs to find the coefficient of weight from the reported local losses for aggregation. This demonstrates FedASL does not add any additional computation compared with standard



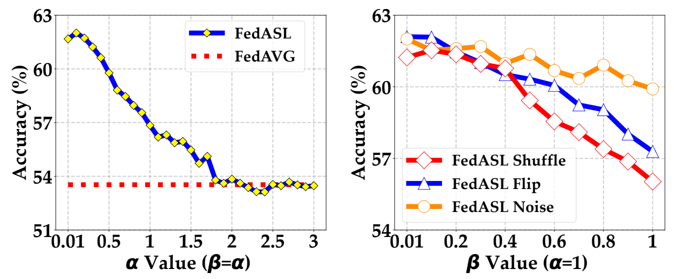
(a) Different algorithm (b) `CIFAR10` model

Fig. 7. Server-side computation requirement for different algorithms. (a) Comparison of computation requirement of FedASL with other algorithms for different datasets. (b) Comparison of computation requirement of FedASL with other algorithms for different models of `CIFAR10` dataset.



(a) `CIFAR10` shuffle (b) `CIFAR10` flip

Fig. 8. Auto weighting by FedASL. (a) Average weight coefficint of good and bad client in `CIFAR10` Shuffling. (b) Average weight coefficient of good and bad client in `CIFAR10` Flipping.



(a) Effect of α (b) Effect of β

Fig. 9. Effects of changing α and β in our algorithm. (a) The effect of changing α is shown for label shuffling for `CIFAR10` dataset. (b) The effect of changing β is shown for label shuffling, label flipping, and noisy data cases for `CIFAR10` dataset.

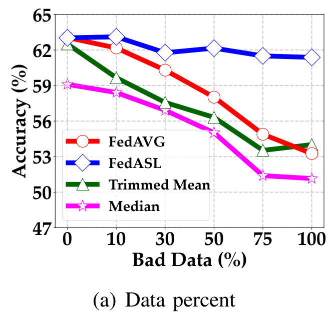
FedAVG. However, for the Trimmed Mean and Median, since they calculate the global model from each parameter of the reported local models, the computation time is very high compared to the FedAVG and FedASL. As we can see in Fig. 7(a), the computation of Trimmed Mean and Median is always higher compared to FedAVG and FedASL even in the small model for `MNIST` and `FEMNIST`. The difference in computation time for the `CIFAR10` model is much higher as we have used CNN model for that. In the CNN model, the number of model parameters is much higher than the simple logistic regression. This adds additional computation to Trimmed Mean and Median algorithms. All the result is shown as normalized to the FedASL algorithm computation time.

Also, to see the behavior of our FedASL when the model size is changed, we compare all the algorithms' computation times in 3 different `CIFAR10` models, i.e. with the 1-VGG layer, 2-VGG layer and 3-VGG layer in the CNN model. The result for different `CIFAR10` model computation costs is shown in Fig. 7(b) where the computation costs are normalized to that of FedASL. We see that FedAVG and FedASL are unaffected by changing model size. In all the 3 cases, they need time since the model parameter has a negligible role in the aggregation techniques for the algorithm. However, for the Trimmed Mean and Median, as they compute the global model based on every model parameter, with the increase of model size, their computation times increase. For VGG-3, the calculation time for Trimmed Mean is around 12 times and Median is around 21 times compared to FedASL. These results show that our algorithm is lightweight and scalable, yet can handle data corruption very well, making it a great choice for various FL scenarios.

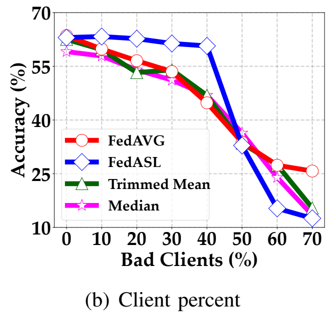
4) How good is FedASL *for low-weighting bad clients?:* To demonstrate how well FedASL can identify the bad clients and assign them lower weights, we track the average model aggregation weight, A_k, of good clients and bad clients for `CIFAR10`'s shuffling and flipping cases. In each round of training, 30% of clients are selected randomly. Fig 8(a) shows the weight coefficient in shuffling case and Fig 8(b) shows the flipping case. The result shows that throughout the training rounds our algorithm was able to give more weight to the good clients and low weight to the bad clients. So FedASL conforms to our assumptions and auto-weights clients based on their data quality over the FL training rounds.

5) Impact of α: To demonstrate the effect of changing the value of α on the robustness of FedASL, we run an experiment on `CIFAR10` with 30% corrupted clients and apply shuffling to corrupt the clients. Fig 9(a) shows FedASL's accuracy as we change the value of α from 0.01 to 3. For this experiment, we set $\beta = \alpha$. We see that when we use smaller values of α resulting in smaller good regions, the accuracy is high. This is because, with a narrow good region, most of the clients within that region are more likely to be good clients. However, when we choose an extremely small α (e.g., 0.01), the good region is only 0.02σ wide and holds too few clients, and does not represent the distribution of the overall system well.

6) Impact of β: To demonstrate the effect of changing the value of β on the robustness of FedASL, we run an experiment on `CIFAR10` with 30% corrupted clients and apply shuffling, flipping, and noising to corrupt the clients. Fig. 9(b) shows the result for changing the value of β. In this experiment keeping the $\alpha = 1\sigma$, we change β value from 0.01 to 1. The β value controls the client's weight for being in the good region. We see an interesting trend for FedASL in shuffling case, if we give a very low β value, that means, clients outside the good region have the least effect, but since good clients may have been outside the good region also, excluding them

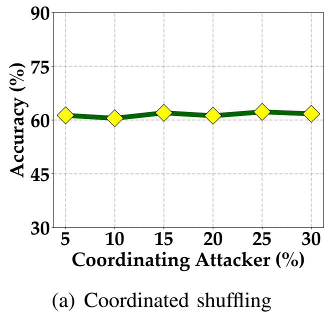


(a) Data percent

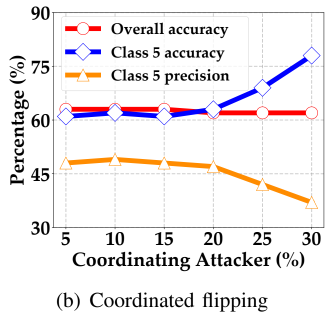


(b) Client percent

Fig. 10. Effects of data corruption and percentage of client corruption in different algorithms for `CIFAR10` dataset.



(a) Coordinated shuffling



(b) Coordinated flipping

Fig. 11. Performance of FedASL with up to 30% coordinated bad clients for `CIFAR10`. (a) Coordinated shuffling with similar label alteration. (b) Coordinated flipping with all bad clients flipping their label to class 5.

completely decreased the overall accuracy. But after increasing β value more, there is an optimum point where the effect of good and bad clients that are outside the good region have an optimum contribution to the global model. Beyond that point, if we increase the β value, that means we are giving more weights to the outside clients where most of them are bad, and decrease the overall global accuracy. The same trend can be seen in the FedASL flip case. Initially, with increasing β value, the accuracy increases but after crossing the optimum point, the accuracy starts to decrease as well. For the FedASL noise cases, the pattern is similar but has some random increase in accuracy. The reason is that there are more good clients outside of the good region as well as more bad clients inside the good region in case of noisy data. So, selecting β properly will have a great effect on attaining good accuracy in our model.

7) Impact of percentage of bad data: To demonstrate the effect of the percentage of bad data on the robustness of FedASL, we run an experiment on `CIFAR10` with 30% corrupted clients with different levels of corrupted data. Fig. 10(a) represents the result for the effect of the percentage of bad data. We see that the accuracy of the existing algorithms decreases with the increase of the percentage of bad data in the corrupted clients. But for the FedASL, with the increase of bad data, the accuracy remains almost the same. This means our algorithm can successfully identify the clients with bad data. We see a linear degradation of accuracy for the FedAVG algorithm which suggest that with the increase of bad data percentage, performance of FedAVG decreases. For Trimmed Mean and Median algorithm, the accuracy remains the same for 75% to 100% bad data levels. This analysis suggests that even in the case of low data corruption, FedASL performs better than the existing state of the arts.

8) Impact of percentage of bad clients: To demonstrate the effect of the percentage of bad data on the robustness FedASL, we run an experiment on `CIFAR10` with different levels of corrupted clients and apply shuffling to corrupt the clients. Fig. 10(b) represents the result of the effect of the percentage of bad clients for different algorithms. As we can see in the figure, with the increase of bad clients percentage in the system, the accuracy of the existing algorithms degrades. For our FedASL, it can effectively handle up to 50% of bad clients situation. As we used the statistics of the clients to determine the good and bad clients, when the percentage of bad clients is more than 50%, bad clients start to dominate. It is obvious that any statistics-based approach would fail. But below 50% situation, our FedASL can effectively gain higher performance compared to the existing state-of-the-art techniques. There is an interesting outcome that we get from the analysis, even though Trimmed Mean and Median is said to be Byzantine robust in case of bad clients, they perform worse than FedAVG with a higher percentage of bad clients. We found that after 50% bad data all the models diverged. In a typical scenario, we expect the number of bad clients should be less than 50%. In that case, FedASL is the best choice for Federated learning.

9) Coordinated bad clients: To demonstrate how FedASL can handle coordinated bad clients where the goal is to divert the global model, we consider two scenarios - coordinated shuffling and coordinated flipping. For coordinated shuffling, we consider all bad clients to make the same alteration to their labels. For example, in `CIFAR10` a coordinated shuffling causes all bad clients to change their label ”1” to ”9”. For coordinated flipping, we consider bad clients to change all of their labels to a single label to bias the global model to a single class heavily. For example, in `CIFAR10` a coordinated flipping causes all bad clients to change all their labels to ”5”. The coordinated flipping is stronger coordination than coordinated shuffling as it targets a single class.

As shown in Fig. 11(a) FedASL can successfully handle coordinated shuffling for various degrees of coordinated bad clients. The overall accuracy is more than 60% which indicates that it is close to the accuracy of the non-coordinated case. Fig. 11(b) demonstrates that our model can also handle the coordinated label flipping as the overall accuracy is not affected. We observe that the accuracy of the target label (Class 5 in `CIFAR10` in this experiment) for coordinated flipping has an increasing accuracy as we increase the percentage of coordination. This is because, with coordinated flipping, the global model gets biased towards class 5 and predicts more instances as class 5, capturing classification cases missed before. However, as shown in Fig. 11(b), this increase in accuracy comes at the cost of decreased precision.

V. Related Work

Federated learning is a collaborative machine learning method to learn a model without collecting data from users [23], [24]. In this paper, we focus on horizontal federated learning where each client has data with the same features as others but has personal data. In Federated Averaging(FedAVG), introduced in 2017, a subset of clients is selected per round of training for local training and the weighted average of those trained models is sent back again to some other subset of clients for further training. Thus over multiple rounds of communication, a global model is learned [5]. But since the model is trained from the data from remote clients, the data quality is not ensured and due to the aging and quality of different sensors, the data quality can be degraded. The performance of FL degrades in presence of corrupted clients [6]. And it is well studied that FedAVG can not handle the presence of corrupted clients as it has no mechanism to mitigate the effects of bad clients [6], [7]. To defend against corrupted clients, various algorithms have been proposed in the literature for attaining robust federated learning [11], [20], [25]. Among these techniques some of the statistics-based approaches are notable. One of the solutions is Krum [12] which selects one of the local models as the global model which is similar to the other models. The similarity is measured by finding the Euclidean distance between all the models. Since Euclidean distance can be largely influenced by a single model parameter, Bulyan [8] was proposed which combines Krum and a variant of Trimmed Mean [9] to solve the issue. But Bulyan is not also scalable and computation hungry as it needs to compute Krum as well as Trimmed Mean in every round of training. Among other byzantine robust algorithms, Trimmed Mean [9] takes each model independently to sort and remove some outliers and find the average of the remaining model parameter as the global model parameter. Another way of work is Median [9] where like Trimmed Mean needs each of the model parameters to be considered independently and the median of those parameters is taken as the global model. Another line of work uses Geometric Median(GM) [10], [11] to find out the model parameter for the federated learning. The main limitations of those approaches are that they are computationally heavy and not suitable for FL at the Internet edge with resource constraints. Federated reinforcement learning (FRL) based approaches such as FedPG-BR [26] introduced recently offers resilience to random systematic failure, adversarial attacks, and Byzantine faults. But the main limitation of these approaches is that it considers homogeneous learning agents where in practicce most of the FRL agents are heterogeneous.

Another line of work uses client selection techniques which select clients based on the client's loss function. AFL [14] uses a value function that can be evaluated on the client-side and the server uses those valuations and converts them to some probabilities to select in the next iteration. They consider selecting clients with higher loss value as they mimic having more minority data points. Power-of-Choice Selection Strategies [15] is further work on the previous paper that also selects the clients with the higher loss for the next training phase. Client selection approaches need to tag client updates with their client IDs. Therefore, they cannot maintain client anonymity diminishing FL's privacy.

In [3], [25], [27]–[29], a subset of trusted clients are exploited to mitigate the effects of bad clients. In clustered Federated learning, [25] proposed to separate the client population into two groups i.e. benign and corrupted groups based on the cosine similarity between model parameters. Li et al. [3] uses an encoder-decoder-based approach to find out malicious model updates. In FLTrust [27] suggests maintaining a root dataset and server model that collects a clean small training dataset from the clients. Since in a federated learning setup, the credibility of the trusted clients or validation dataset is not guaranteed, and due to communication and privacy constraints, these approaches violate Federated learning protocols.

In contrast to prior works, we propose FedASL which can automatically mitigate the effect of bad clients in every round of training without any trusted clients or clients id. FedASL is robust against various data corruption scenarios. More importantly, FedASL is a lightweight approach and does not add additional computation. FedASL is also suitable even if there is no data corruption and attains similar accuracy as the standard FL algorithms such as FedAVG.

VI. Concluding Remarks

In this paper, we presented FedASL, a novel and lightweight auto-weighted aggregation-based FL technique. FedASL utilizes local training loss of FL clients to identify clients with corrupted data. We developed a novel region-based segregation between clients with different quality of data. Using extensive evaluation with three different datasets under various scenarios we demonstrated the effectiveness of FedASL. We also showed that FedASL adds no additional computation burden on the central model server.

Limitations. While FedASL demonstrates its effectiveness in addressing FL's heterogeneity using efficient auto-weighting, it also has certain limitations. FedASL is evaluated in a cooperative setting where the clients truthfully report their training loss. Hence, FedASL can be attacked by malicious clients who send fabricated loss values to include their unfavorable model updates in the aggregation. However, the weight regions in FedASL are not disclosed to the FL clients. Also, by tightening the good region, we can reduce the effectiveness of these attacks. FedASL's design is not robust in identifying good and bad user data sources. For instance, a user with biased but good data may fall outside the good region and treated as a user with data corruption. We have not addressed the strong class imbalance in user data which may lead to weak statistical coherence between clients where FedASL cannot effectively identify its weight regions. While not applicable for all scenarios, FedASL offers a novel lightweight auto-weighting approach that can handle data heterogeneity with minimal additional computation.

References

[1] D. R. Brendan McMahan, "Federated learning: Collaborative machine learning without centralized training data." https://ai.googleblog.com/2017/04/federated-learning-collaborative.htm.

[2] K. Hao, "How apple personalizes siri without hoovering up your data." https://www.technologyreview.com/2019/12/11/131629/apple-ai-personalizes-siri-federated-learning.

[3] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen, "Learning to detect malicious clients for robust federated learning," *arXiv preprint arXiv:2002.00211*, 2020.

[4] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, "Federated learning for healthcare informatics," *Journal of Healthcare Informatics Research*, 2021.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[6] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *European Symposium on Research in Computer Security*, pp. 480–501, Springer, 2020.

[7] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to {Byzantine-Robust} federated learning," in *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1605–1622, 2020.

[8] R. Guerraoui, S. Rouault, *et al.*, "The hidden vulnerability of distributed learning in byzantium," in *International Conference on Machine Learning*, pp. 3521–3530, PMLR, 2018.

[9] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*, pp. 5650–5659, PMLR, 2018.

[10] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, pp. 1–25, 2017.

[11] K. Pillutla, S. M. Kakade, and Z. Harchaoui, "Robust aggregation for federated learning," *IEEE Transactions on Signal Processing*, 2022.

[12] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[13] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC*, IEEE, 2020.

[14] J. Goetz, K. Malik, D. Bui, S. Moon, H. Liu, and A. Kumar, "Active federated learning," *arXiv preprint arXiv:1909.12641*, 2019.

[15] Y. J. Cho, J. Wang, and G. Joshi, "Client selection in federated learning: Convergence analysis and power-of-choice selection strategies," *arXiv preprint arXiv:2010.01243*, 2020.

[16] X. Liu and E. Ngai, "Gaussian process learning for distributed sensor networks under false data injection attacks," in *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, pp. 1–6, IEEE, 2019.

[17] I. Dietrich and F. Dressler, "On the lifetime of wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 1, pp. 1–39, 2009.

[18] P. Wais, S. Lingamneni, D. Cook, J. Fennell, B. Goldenberg, D. Lubarov, D. Marin, and H. Simons, "Towards building a high-quality workforce with mechanical turk," *Proceedings of computational social science and the wisdom of crowds (NIPS)*, pp. 1–5, 2010.

[19] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[20] J. So, B. Güler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2168–2181, 2020.

[21] 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.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[23] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[24] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[25] F. Sattler, K.-R. Müller, T. Wiegand, and W. Samek, "On the byzantine robustness of clustered federated learning," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8861–8865, IEEE, 2020.

[26] X. Fan, Y. Ma, Z. Dai, W. Jing, C. Tan, and B. K. H. Low, "Fault-tolerant federated reinforcement learning with theoretical guarantee," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[27] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," in *ISOC Network and Distributed System Security Symposium (NDSS)*, 2021.

[28] Y. Han and X. Zhang, "Robust federated learning via collaborative machine teaching," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 4075–4082, 2020.

[29] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19586–19597, 2020.