Privacy-Preserving Distributed Machine Learning based on Secret Sharing

Ye Dong[1,2], Xiaojun Chen[1(✉)], and Liyan Shen[1,2]

[1] Institute of Information Engineering,Chinese Academy of Sciences,Beijing,China
[2] School of Cyber Security, University of Chinese Academy of Sciences,Beijing,China
{dongye,chenxiaojun,shenliyan}@iie.ac.cn

Abstract. Machine Learning has been widely applied in practice, such as disease diagnosis, target detection. Commonly, a good model relies on massive training data collected from different sources. However, the collected data might expose sensitive information. To solve the problem, researchers have proposed many excellent methods that combine machine learning with privacy protection technologies, such as secure multiparty computation(MPC), homomorphic encryption(HE), and differential privacy. In the meanwhile, some other researchers proposed distributed machine learning which allows the clients to store their data locally but train a model collaboratively. The first kind of methods focuses on security, but the performance and accuracy remain to be improved, while the second provides higher accuracy and better performance but weaker security, for instance, the adversary can launch membership attacks from the gradients' updates in plaintext.
In this paper, we join secret sharing to distributed machine learning to achieve reliable performance, accuracy, and high-level security. Next, we design, implement, and evaluate a practical system to jointly learn an accurate model under semi-honest and servers-only malicious adversary security, respectively. And the experiments show our protocols achieve the best overall performance as well.

Keywords: Secret Sharing · Distributed Machine Learning · Privacy-Preserving.

1 Introduction

Recent advances in machine learning have produced exciting achievements both in academia and industry, the machine learning systems are approaching or even surpassing human-level accuracy in speech, image and text recognition. That thanks to algorithmic breakthroughs and hardware developments, which help our systems process massive amounts of data.

However, massive data collection, which is a key step in learning an accurate model, has caused public panic about privacy breaches. As a consequence, the useful but sensitive data such as medical records, are forbidden to be shared among different institutes due to proprietary reasons or compliance requirements[21,33]. Privacy-preserving machine learning via MPC provides a promising solution by allowing different institutions to train various models based on their joint data without revealing any sensitive information beyond the outcome.

The state-of-the-art solutions for privacy-preserving machine learning based on MPC, i.e. [23,27,29], are many orders of magnitude slower than training on plaintext. The main source of inefficiency is that the bulk of computation in the training phase takes place within a secure manner such as garbled circuits or HE. It is well-known that computing complex function, especially non-linear function, in secure form is very expensive.

To improve the efficiency, we design our protocols based on distributed machine learning and instead of joining the secure computation technologies to these expensive computations in our protocols, we join secret sharing to the gradients' sharing phase which only consists of simple arithmetics, i.e. addition. In this way, we can improve efficiency greatly while still meet the security requirements. Finally, each client can learn no information beyond the trained model, and the parameter servers can learn no sensitive information.

1.1 Our Contribution

In this paper, We design two new and efficient protocols for privacy-preserving linear regression,Multilayer perceptron(MLP), and Convolutional neural network(CNN) in the distributed machine learning settings assuming the data are distributed across the clients. Then we give the security analysis of our protocols. And lastly, we implement, evaluate our protocols and compare them with other latest results in a comparable environment.

Resistance to semi-honest and servers-only malicious adversary. In the semi-honest setting, we use Shamir's Secret Sharing to design protocol $\mathit{\Gamma}_{sash}$. As long as no more than $t-1$(t is the threshold) servers can collude and at least two clients are honest, $\mathit{\Gamma}_{sash}$ can against users-only, servers-only and users-servers threat models.

In the malicious setting, to resist servers-only malicious modifications, we design a verifiable protocol $\mathit{\Gamma}_{sam}$ via a variant of Secret Sharing with information-theoretic Message Authentication Code(MAC)[12]. In $\mathit{\Gamma}_{sam}$, besides sharing the gradients $\boldsymbol{g}$, each client computes $\boldsymbol{g}$'s information-theoretic MAC and shares it among parameter servers. In addition to protecting privacy like $\mathit{\Gamma}_{sash}$, $\mathit{\Gamma}_{sam}$ can prevent servers from malicious modifications as long as no more than $t-1$ servers can collude and no client colludes with servers.

Performances Our privacy-preserving machine learning protocols are more efficient than state-of-the-art solutions. For example, for a dataset with 60,000 samples and 784 features, our protocol is $30\times$ faster than the protocols implemented in[28] for CNN in the semi-honest setting. And even our malicious protocol can achieve $25\times$ improvement.

As discussed above, our protocols can be divided into two phases, offline and online phase. In a comparable experimental environment, our protocols are at the same efficiency level with[25] in the online phase, but we can achieve 5×-10× improvement in the offline phase. Note that vectorization, i.e. operating on matrices and vectors, is critical in the efficiency of training on plaintext, we can benefit from this technique here. For instance, we find that the vectorized protocols improve our efficiency around 7.5-10× in the online phases and 5-10× in the offline phases.

As experiments show, our protocols are even more competitive with training on plaintext. For instance, for the MNIST[1], our protocols can achieve the same level of accuracy at a total time of 40.2 seconds for linear regression, 205.2 seconds for MLP and 723.6 seconds for CNN.

1.2 Related Work

In the earlier stage, the work on privacy-preserving machine learning mainly focused on traditional machine learning models such as linear regression[3,4,5], logistic regression[8], decision trees[2], k-means clustering[6,9] and SVM[7,10]. These papers proposed solutions based on MPC but were limited to a particular kind of model. For example, Nikolaenko et. al.[13] and Gascon et. al.[20] presented secure computation protocols for linear regression on mega datasets via leveled-HE(LHE) and garbled circuits. However, both papers are limited to linear regression and the key problems are both reduced to solving a linear system using Yao's garbled circuits.And the efficiency overheads appear to be very high.

For the logistic regression, Wu et. al.[14] chose to approximate the sigmoid function using polynomials and train the model using LHE, but the complexity is too high and accuracy is poor. Yupeng Zhang et. al.[23] presented a solution that can be applied to linear regression, logistic regression and neural networks on the two-server setting where data owners distribute their private data among two non-collude servers and proposed a new method based on piecewise linear function for the non-linear function to improve efficiency. However, there is still a big gap compared to training on plaintext.

Meanwhile, Jian L. et. al.[24] proposed a secure inference framework that can protect the server's model and client's data at the same time. Privacy-preserving predictions were also studied by Galad-Bachrach et. al.[22,32]. But the models in this setting need to be trained on plaintext ahead of time.

Shokri and Shmatikov[17] proposed a solution by sharing the model's gradients among the clients during the training via a parameter server. They improve the efficiency greatly, but the leaks of gradients could weaken the security[30]. Le Trieu Phon et. al.[28] extended the result against semi-honest adversaries using LHE, but the performance is too poor. Recently, Bonawitz et. al.[25] proposed a protocol to secure aggregate gradients. But their offline phase is very complex, and if some clients dropped out, the efficiency would be reduced.

[1]MNIST database, `http://yann.lecun.com/exdb/mnist/`. Accessed: 2017-09-24.

So it is urgent and challenging to propose a secure framework to train complex and huge machine learning models efficiently.

An orthogonal and complementary work considers the differential privacy of machine learning algorithms[15,19]. In this setting, the key point is to introduce an additive noise to the data or the update function, so as to prevent the adversary from inferring the data from the released model. Our system can be combined with such technology to provide stronger security.

1.3 Roadmap

In section 2, we introduce the preliminaries. In section 3, we give our system architecture and the design of our protocols. We also analyze the correctness and privacy of our protocols. In section 4, we give the results of our experiments, and in section 5, we conclude our paper.

2 Preliminaries

In this paper, the notations we used are as below:

a denotes a scalar, $\boldsymbol{g}$ denotes a vector and g_i denotes the i-th element of $\boldsymbol{g}$. $\mathbf{X}$ denotes a matrix and $\mathbf{X}_{ij}$ denotes the element in row i and column j. $\langle a \rangle_i$ denotes the i-th shares of a, and the same as to vector and matrix. And N denotes the number of servers and M denotes the number of clients.

2.1 Machine Learning

In this section, we briefly review the distributed machine learning and some machine learning algorithms: linear regression, MLP, CNN. All these algorithms are classic and can be found in standard machine learning papers or textbooks.

Distributed Machine Learning is a new kind of settings of machine learning, there exist many different settings[31],[26] and we adopt the Parameter-Server(PS) setting[16] in this paper. As shown in Fig 1, in the P-S setting there are many clients and one parameter server. Each client has a private dataset. Note that the data between different clients are of the same type, i.e. medical records. In general, there are five phases in the P-S setting, local training phase, upload phase, aggregate phase, download phase, and update phase.

Before training, all the clients negotiate a unified model and everyone stores a replica and initializes it. In the local training phase, each client trains the model locally and computes the gradients $\boldsymbol{g} = (g_0, g_1, ..., g_{n-1})$, where g_i is the gradient for coefficient w_i. Next, the clients will upload $\boldsymbol{g}$ to the parameter server. On the other hand, the parameter server will wait a while to receive enough gradients and then aggregate them as sum $\boldsymbol{g}_s = \sum \boldsymbol{g}$ or average $\boldsymbol{g}_{avg} = \frac{1}{M}\boldsymbol{g}_s$. Finally, the clients will download the aggregated gradients and update the local model for the next training epoch.

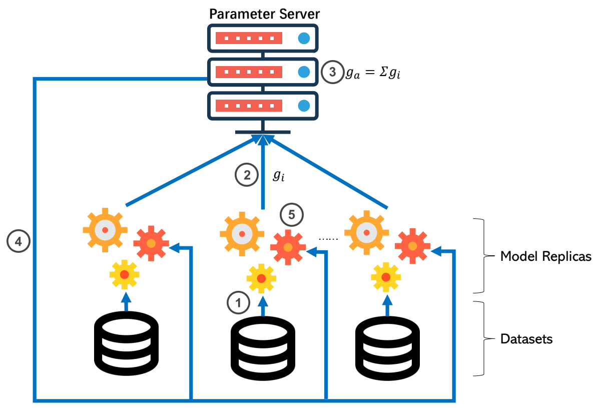


Fig. 1. Distributed Machine Learning, Parameter-Server setting. ① local training, ② upload gradients, ③ aggregate gradients, ④ download gradients, ⑤ update the local model.

Distributed Selective SGD Learning the parameters of a model is not easy, especially for a complex model, i.e. neural networks. The methods that solve this problem are typically variants of gradient descent algorithm(GD)[11]. Among these algorithms, stochastic gradient descent(SGD) is a drastic simplification that computes the gradients over a subset(mini-batch) of the whole dataset while maintains high accuracy.

Let $\boldsymbol{w}$ be the vector of all parameters in a model, w_i is the i-th element of $\boldsymbol{w}$. Let E be the error function which can be based on L^2 norm or cross-entropy. The update rule of SGD for a parameter w_i is

$$w_i = w_i - \alpha \frac{\partial E_i}{\partial w_i} \tag{1}$$

where α is the learning rate and E_i is the error computed over the mini-batch i.

Note that the update of each parameter is independent, so that the client can send a portion of gradients which are important instead of all gradients to reduce communication[17], which we use in this paper.

Linear Regression Given n training data samples $\boldsymbol{x}_i$, each contains d features and the corresponding labels y_i, where $y_i = \pm 1$. Training a linear regression model is a process to learn a function f such that $f(\boldsymbol{x}_i) = y_i$. Linear regression has many applications in real life, i.e. detecting diseases in medical research.

In linear regression, the function f is a linear operation and can be represented as the inner product of $\boldsymbol{x}_i$ and the coefficient vector $\boldsymbol{w}$:

$$f(\boldsymbol{x}_i) = \sum_{j=1}^{d} x_{ij} w_j = \boldsymbol{x}_i \cdot \boldsymbol{w} \tag{2}$$

where $\cdot$ denote the inner product of two vectors.

Multi-Layer Perceptron Deep Learning aims to extract more complex features than traditional machine learning models. MLP is one basic form of deep learning models.

Fig 2(a) shows a typical MLP with two hidden layers, each node represents a neuron, it receives the output of the neurons of previous layers plus a bias from a special neuron. Then it computes a weighted average of its inputs. Finally, the neuron applies a non-linear function to the weighted average.

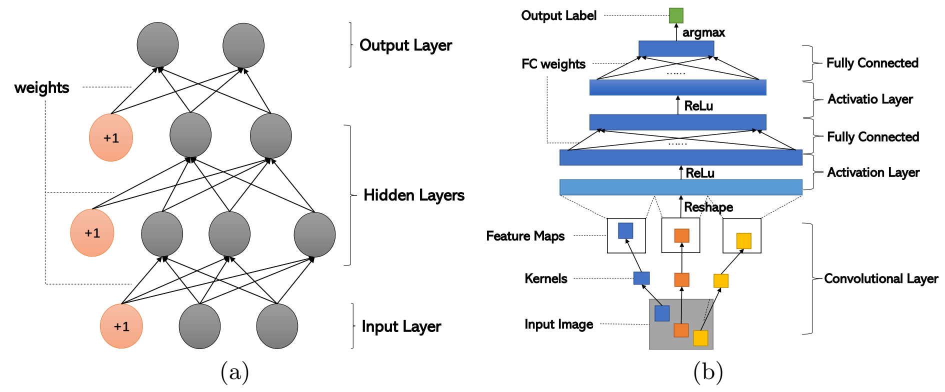


Fig. 2. Neural Networks. (a) is for MLP, (b) is for CNN.

Convolutional Neural Networks CNN has gained much more attention in the past decades owing to its superb accuracy. While there are many different CNNs, they all share a similar structure.

As shown in Fig 2(b), the input to a CNN is represented as a matrix $\mathbf{X}$ where each element corresponds to the value of a pixel. Pictures can have multiple color channels, i.e. RGB, in which case the picture is represented as a multidimensional matrix, i.e. tensor. Compared to MLP, CNN has additive layers: (i)Convolution layer,(ii) (Mean or Max)-Pooling layer, both play important roles.

2.2 Secure Computation

Secret-Sharing. Shamir's Secret Sharing[1] is a powerful cryptographic primitive which allows a client to split a secret into n shares, so that any less than t shares reveal no information about the original secret while any t shares can recover the secret.

Definition 1. *Shamir's Secret Sharing scheme consists of a sharing algorithm $\mathcal{S}$ and a reconstruct algorithm $\mathcal{R}$.*

$\mathcal{S}$ takes a secret s, a threshold t, and n as inputs, outputs n shares of s

$$\mathcal{S}(s,t,n) \rightarrow \{\langle s\rangle_0, \langle s\rangle_1, ..., \langle s\rangle_n\} \tag{3}$$

with $t \leq n$.

$\mathcal{R}$ takes the threshold t, a subset of the shares with size m as inputs, and recovers the secret s

$$\mathcal{R}(\{\langle s\rangle_0, \langle s\rangle_1, ..., \langle s\rangle_m\}, t) \rightarrow s \tag{4}$$

with $m \geq t$.

Correctness requires that $\forall s \in \mathbb{F}$, $\forall t$, $n(t \leq n)$, if $\{\langle s\rangle_0, \langle s\rangle_1, ..., \langle s\rangle_n\}$ are shares of s, then any subsets of size $m(m \geq t)$ could reconstruct the original secret s.

Security requires that any subsets of shares of size $m'(m' \leq t-1)$ disclose no information about s, which means that $\forall s, s' \in \mathbb{F}$ and two subsets of shares with size $m'(m' \leq t-1)$, no polynomial-time adversary $\mathcal{A}$ can distinguish the distribution of the two subsets

$$|Pr[\mathcal{A}(\{\langle s\rangle_0, ..., \langle s\rangle_{m'}\}) = 1] - Pr[\mathcal{A}(\{\langle s'\rangle_0, ..., \langle s'\rangle_m\}) = 1]| \leq \frac{1}{p(x)} \tag{5}$$

where $p(\cdot)$ is a positive polynomial and x is sufficiently large.

Secret Sharing with information-theoretic MAC. The Shamir's Secret Sharing scheme can only against semi-honest adversaries, but not resist malicious modifications. So we import a variant of information-theoretic Message Authentication Code(MAC)[12].

Definition 2. *Secret Sharing scheme with information-theoretic MAC consists of a sharing algorithm $\mathcal{S}$, a reconstruct algorithm $\mathcal{R}$, an authentication function $\boldsymbol{\delta}$, a verification function $\boldsymbol{\upsilon}$, and a global key α.*

$\mathcal{S}$ takes a secret s, the function $\boldsymbol{\delta}$, the threshold t, and n as inputs, and outputs their shares

$$\mathcal{S}(s, \boldsymbol{\delta}, t, n) \rightarrow \{(\langle s\rangle_0, ..., \langle s\rangle_n), (\langle \boldsymbol{\delta}(s)\rangle_0, ..., \langle \boldsymbol{\delta}(s)\rangle_n)\} \tag{6}$$

$\mathcal{R}$ takes the threshold t, the function $\boldsymbol{\upsilon}$ and the subsets of shares as inputs, and outputs s

$$\mathcal{R}((\{\langle s\rangle_0, ..., \langle s\rangle_m\}), (\{\langle \boldsymbol{\delta}(s)\rangle_0, ..., \langle \boldsymbol{\delta}(s)\rangle_m\}), \boldsymbol{\upsilon}, t) \rightarrow s \tag{7}$$

if $\boldsymbol{\upsilon}(s, \boldsymbol{\delta}(s), \alpha) = 1$, else $\mathcal{R}(\cdot)$ returns $\perp$.

Note $\boldsymbol{\delta}(s) = \alpha \cdot s \mod p$ and $\boldsymbol{\upsilon}(\cdot) = 1$ if and only if the reconstructed s and $\boldsymbol{\delta}(s)$ satisfy $\boldsymbol{\delta}(s) = \alpha \cdot s \mod p$.

Correctness and privacy against the semi-honest adversary are identical to what we have mentioned in Shamir's Secret Sharing scheme.

Since we require that the global key α is unknown to the adversary $\mathcal{M}$. So $\forall \langle s\rangle_i$, even $\mathcal{M}$ modifies it with only one bit, the possibility that $\mathcal{M}$ can construct a valid share of its MAC is negligible.

3 System Architecture

Our scheme is based on distributed machine learning except that we introduce two or more non-collude servers like[23,29], to protect the privacy. However, the servers in our protocols are not responsible for storing data or training the model but aggregating the gradients.

Instead of uploading the plain gradients to one parameter server in distributed machine learning, the clients share gradients after each training epoch and upload these shares to corresponding servers in our protocols.

In this paper, we propose two protocols, Γ_{sash} and Γ_{sam}. In Γ_{sash}, we require the servers are semi-honest or honest-but-curious and at most $t-1$ servers can collude. In Γ_{sam}, we enhance our security capabilities to against servers-only malicious adversary via Secret Sharing with information-theoretic MAC.

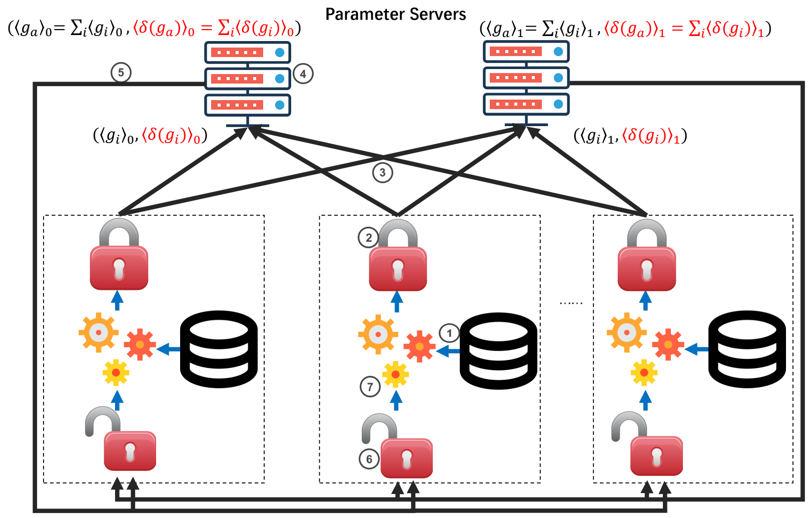

Parameter Servers
$(\langle g_a\rangle_0 = \sum_i \langle g_i\rangle_0, \langle\delta(g_a)\rangle_0 = \sum_i\langle\delta(g_i)\rangle_0)$
$(\langle g_a\rangle_1 = \sum_i \langle g_i\rangle_1, \langle\delta(g_a)\rangle_1 = \sum_i\langle\delta(g_i)\rangle_1)$
$(\langle g_i\rangle_0, \langle\delta(g_i)\rangle_0)$
$(\langle g_i\rangle_1, \langle\delta(g_i)\rangle_1)$
5
4
3
2
1
7
6
......

Fig. 3. Architecture for our protocols with two parameter servers.①local training, ②share the gradients(and MACs), ③upload gradients' shares(and MACs' shares), ④aggregate gradients' shares(and MACs' shares), ⑤download aggregated gradients' shares(and MACs' shares), ⑥reconstruct aggregated gradients(, MACs and verify), ⑦update the local model and train it again(or abort). Note that MACs are for malicious security.

3.1 Γ_{sash}—Protocol for Semi-honest Security

In the offline phase, the clients communicate with each other to agree on a common secret sharing scheme, a machine learning model, i.e. a neural network, through secure channels. Also, clients need to generate random numbers independently. The parameter servers initialize the shares of $\boldsymbol{g}_a$ as all zeros. Then each client establishes a TLS/SSL secure channel with each server to protect the integrity of the shares.

Next, each client trains the model using private data locally and computes the gradients $\boldsymbol{g}$ as Equation 1.

The elements of $\boldsymbol{g}$ are all float-point decimal numbers, which are not suitable for arithmetic operations in secure computations. So we have to encode all the elements of $\boldsymbol{g}$ as integers in a large finite field.

For instance, we can multiply them by a large scaling factor and truncate the results as integers modulo p, p is a big prime. When decode, we could determine the sign of an encoded element and divide the scaling factor[2]. As shown in the experiments, the errors introduced by truncation are so small that have little impacts on the final model.

Note that the clients can only encode and share a portion of gradients which are important, and we use $\boldsymbol{s}_i^{(up)}$ to indicate them.

Next, the client i shares $\boldsymbol{g}_i$ as $\langle \boldsymbol{g}_i \rangle_0, \langle \boldsymbol{g}_i \rangle_1, ..., \langle \boldsymbol{g}_i \rangle_{N-1}$, sends $\langle \boldsymbol{g}_i \rangle_j$ and $\boldsymbol{s}_i^{(up)}$ to the j-th server. Note that both encoding and secret sharing can be accelerated via vectorization.

On the other hand, the servers would wait a while to receive enough secret shares, i.e. all the secret shares, and then compute the secret shares of the aggregated gradients according to $\{\boldsymbol{s}_i^{(up)}\}$

$$\langle \boldsymbol{g}_a \rangle_j = \sum_i \langle \boldsymbol{g}_i \rangle_j \tag{8}$$

After the aggregation, the servers will reply to clients' requests $\boldsymbol{s}_i^{(down)}$ with the aggregated shares, $\boldsymbol{s}_i^{(down)}$ indicates the elements to be downloaded. The details are in Fig 4. We will prove the correctness and security below.

Correctness. In Γ_{sash}, the parameter servers are responsible for aggregating the gradients' shares, which is in secret sharing form. To prove our protocol's correctness, we import *lemma* 1, which we prove in **Appendix A.1**.

Lemma 1. *The addition of secret shares is the secret shares of the sum. For example,* $\{\langle \boldsymbol{x} \rangle_i\}$ *are the secret shares of* $\boldsymbol{x}$ *and* $\{\langle \boldsymbol{y} \rangle_i\}$ *are the secret shares of* $\boldsymbol{y}$*, then* $\{\langle \boldsymbol{z} \rangle_i = \langle \boldsymbol{x} \rangle_i + \langle \boldsymbol{y} \rangle_i\}$ *are the secret shares of* $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}$.

According to *lemma* 1, we know that each server aggregates one of the secret shares of $\boldsymbol{g}_a$ rightly. So in the end, the clients will receive enough secret shares and reconstruct the $\boldsymbol{g}_a$.

Security. In Γ_{sash}, we consider the security of the training protocol against semi-honest adversaries with three different threat models, namely users-only threat model where some users are corrupted, servers-only threat model where some servers are corrupted, and users-servers threat model where some users and servers are controlled by adversary.

[2]https://mortendahl.github.io/2017/04/17/private-deep-learning-with-mpc/

Theorem 1 (Privacy in Semi-honest Adversary). *The protocol $\mathit{\Gamma}_{sash}$ is secure in presence of semi-honest adversaries, meaning they leak no sensitive information about the honest clients' gradients, as long as the adversary can only corrupt no more than $t-1$ servers and $M-2$ clients.*

In the users-only threat model, what the adversary $\mathcal{A}$ can get about honest clients is the sum of their gradients.

$$\sum_{j\in honest.} \boldsymbol{g}_j = \sum_{i\in all} \boldsymbol{g}_i - \sum_{k\in corrupted} \boldsymbol{g}_k \tag{9}$$

Even the number of corrupted clients are $M-2$, $\mathcal{A}$ get no information about the gradients towards a particular client. So we can protect the honest clients' privacy.

In the servers-only threat model, we require $\mathcal{A}$ can only corrupt up to $t-1$ servers. So $\mathcal{A}$ can not distinguish the gradients' shares from pseudorandom numbers under Shamir's Secret Sharing scheme, which means $\mathcal{A}$ can not violate clients' privacy.

From users-only threat model and servers-only threat model, we can know that even $\mathcal{A}$ corrupt $t-1$ servers and $M-2$ clients, what $\mathcal{A}$ can get is as in the users-only threat model since the corrupted servers leak no private information.

3.2 $\boldsymbol{\mathit{\Gamma}_{sam}}$—Protocol for Servers-only Malicious Security

Protocol $\mathit{\Gamma}_{sash}$ can against semi-honest adversary, but not ensure that servers will sum up the gradients honestly, which means if a server is controlled by a malicious adversary $\mathcal{M}$, he can launch an attack,i.e. poisoning attack. For instance, $\mathcal{M}$ can replace the gradients' shares with random values to reduce the performance of the final model.

In order to prevent this kind of attacks, we propose $\mathit{\Gamma}_{sam}$, a protocol based on Secret Sharing with information-theoretic MAC, which can detect servers' malicious behaviors. So in $\mathit{\Gamma}_{sam}$, in addition to sharing the gradients, the clients have to compute and share the gradients' MAC $\boldsymbol{\delta}(\boldsymbol{g})$. And the servers have to aggregate the shares of $\boldsymbol{\delta}(\boldsymbol{g})$. The details are in Fig 4.

Correctness. In protocol $\mathit{\Gamma}_{sash}$ we have proved that the sum of secret shares is the secret shares of the sum. Now we bring in *lemma* 2, we give its proof in **Appendix A.2**.

Lemma 2. *The sum of homomorphic MACs is the MAC of the sum. For example, $\boldsymbol{\delta}(\boldsymbol{x})$ and $\boldsymbol{\delta}(\boldsymbol{y})$ are homomorphic MACs of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively, then $\boldsymbol{\delta}(\boldsymbol{x}) + \boldsymbol{\delta}(\boldsymbol{y})$ is the homomorphic MAC of $\boldsymbol{x}+\boldsymbol{y}$.*

As long as the clients compute homomorphic MACs of gradients, we have that the sum of the MACs is the MAC of the sum of gradients on the basis of *lemma* 2. So combining with *lemma* 1, we can prove the correctness of $\mathit{\Gamma}_{sam}$.

Security. The privacy of $\varGamma_{sam}$ is the same as $\varGamma_{sash}$, and we can prevent the corrupted servers from malicious modification with a servers-only threat model.

Theorem 2 (Resistance to Malicious Adversary in servers-only threat model). *The protocol $\varGamma_{sam}$ is secure against malicious adversaries in the servers-only threat model, meaning the adversary can not launch active poisoning attacks without detection, as long as the adversary can only corrupt no more than $t-1$ servers.*

As long as the global secret key α is only known to clients, the possibility that $\mathcal{M}$ can construct a valid MAC for an arbitrary secret s is equal to the possibility that $\mathcal{M}$ can get α. So we have

$$Pr(\mathcal{M}(\boldsymbol{\delta}, s) = 1) = \frac{1}{2^{\lceil \log p \rceil}} \tag{10}$$

As long as the prime p is sufficient large, the possibility is negligible.

Assuming $\mathcal{M}$ modifies the $\langle \boldsymbol{g} \rangle_u$ to launch a poisoning attack. So he would add the modified shares to aggregated shares, which denoted as $\langle \boldsymbol{g}_a \rangle_u'$. If $\langle \boldsymbol{g}_a \rangle_u'$ is downloaded and the client would try using it to reconstruct the $\boldsymbol{g}_a$

$$\boldsymbol{g}_a' = \mathcal{R}(\{\langle \boldsymbol{g}_a \rangle_i\}_{i \in \{t\}, i \neq u} \cup \{\langle \boldsymbol{g}_a \rangle_u'\}) \tag{11}$$

Also, the client can reconstruct $\boldsymbol{\delta}(\boldsymbol{g}_a)$ at the same time.

It is obvious that the possibility $\boldsymbol{v}(\alpha, \boldsymbol{g}_a', \boldsymbol{\delta}(\boldsymbol{g}_a)) = 1$ is negligible as long as α is unknown to the $\mathcal{M}$ and no more than $t-1$ servers could collude.

So in $\varGamma_{sam}$, we can avoid disclosing any particular client's gradients under the assumption of Shamir's Secret Sharing scheme. Moreover, we could detect corrupted servers' malicious behaviors via information-theoretic MAC.

```
Secure Training Protocol

Input: $M$ local datasets and replicas of a model
Output: a trained model

if Server then
    Initialize. Initializing $\langle \boldsymbol{g}_a \rangle_j$ and $\langle \boldsymbol{\delta}(\boldsymbol{g})_a \rangle_j$ as all-zeros;
    Update. After receiving gradients' secret shares, MACs' secret shares and $\boldsymbol{s}_i^{(up)}$ from each client, each server will add these secret shares to $\langle \boldsymbol{g}_a \rangle_j$ and $\langle \boldsymbol{\delta}(\boldsymbol{g})_a \rangle_j$ according to $\boldsymbol{s}_i^{(up)}$, usually $\boldsymbol{s}_i^{(up)} \subset \boldsymbol{g}$;
    Response. For each downloading request, each server will reply with the gradients' shares and MACs' shares according to $\boldsymbol{s}_i^{(down)}$;
if Client then
    Initialize. Initializing models according to the same policy;
    Local training. Each client gets a mini-batch from the local dataset, trains the model and computes the gradients $\boldsymbol{g}$;
    Secret Sharing. The clients encode $\boldsymbol{g}$ and generate the secret shares of $\boldsymbol{g}$ as $\{\langle \boldsymbol{g} \rangle_j\}$ , compute $\boldsymbol{\delta}(\boldsymbol{g})$ and share it as $\{\langle \boldsymbol{\delta}(\boldsymbol{g}) \rangle_j\}$ (or only processing the top $k$ of $\boldsymbol{g}$ with the largest absolute value, where we use $\boldsymbol{s}_i^{(up)}$ to indicate the corresponding elements);
    Upload. Sending the gradients' secret shares, MACs' secret shares and $\boldsymbol{s}_i^{(up)}$ to corresponding servers;
    Download. Sending a downloading request along with $\boldsymbol{s}_i^{(down)}$, which indicates the elements to be downloaded, to all servers;
    Reconstruct. After sending the request, the client will wait until receiving enough secret shares to reconstruct the aggregated gradients, aggregated MACs and verify the validity, update the local model and go to Local training or abort if the verification failed;
```

Fig. 4. Details for our protocol. Note that the red-and-underlined parts are only required for malicious protocol Γ_{sam}(not necessary for semi-honest protocol Γ_{sash}).

4 Experiments

4.1 Environment

Our experiments are executed on three Intel(R) Xeon(R) CPU E5-2650 v3@ 2.30GHz servers with each has 64G RAM in the LAN setting.

We simulate 2 parameter servers and 32 clients. All protocols have been implemented in Python3 language, and we use Tensorflow 1.13.1[3] library, this popular machine-learning library has been used by major Internet companies such as Google.

[3]https://github.com/tensorflow/tensorflow/releases/tag/v1.13.1

4.2 Experiments setup

In our experiments, we use MNIST as our training set. And we compare all results with two baseline scenarios. The first is the basic federate learning Γ_{fl}, which consists of one parameter server and 32 clients with no secure techniques. The other scenario is Google's Secure Aggregation protocol Γ_{sag}[25], which masks the gradients before uploading[4]. All the protocols are executed synchronously.

For all the scenarios, we implement linear regression, MLP, and CNN. We compare them in accuracy, convergence rate and performance in detail below.

4.3 Experiments Results

Accuracy We compare the same model in different scenarios, we find that our accuracies in the semi-honest and malicious setting are both nearly to the accuracy of Γ_{fl}. The highest accuracy's drop is within 0.01 for CNN and MLP, and 0.02 for linear regression.

We plot the accuracy changes along with the training epochs. And we show that with the increasing of the training epochs, the influence produced by encoding a float-number as a big integer is being smaller and smaller. For instance, **Fig.5**(a) shows that in a CNN, the curve for Γ_{sash} almost coincides with the curve for Γ_{fl}. And we get similar results for linear regression and MLP, we plot them in **Appendix B** due to the space limit.

The best accuracies for each model in all protocols are shown in Table 1.

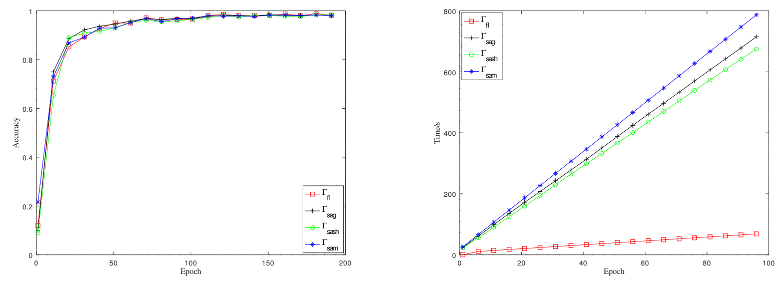
Table 1. Accuracy for each model in all protocols.

Protocol / Model	Γ_{fl}	Γ_{sag}	Γ_{sash}	Γ_{sam}
Linear Regression	0.929	0.924	0.918	0.913
MLP	0.979	0.977	0.978	0.974
CNN	0.997	0.994	0.995	0.991

Convergence Rate Our experiments also illustrate that the convergence rates in all protocols over training epochs are at the same level, the secure techniques do not influence the results much.

For instance, the convergence rates of Γ_{sash} and Γ_{sam} are almost the same as Γ_{fl} for CNN. From Fig 5(a), it is obvious that the models all approach 0.95 at around 50 epochs and reach 0.99 at around 100 epochs in different protocols.

[4]We only implement the basic secure aggregation with no dropouts



(a) (b)

Fig. 5. Experimental results of CNN. (a) is for accuracy, (b) is for performance.

Performance Our protocols can be divided into offline phase and online phase naturally. In the offline phase, the clients mainly generate random numbers independently. We run the process 10 times for each model and take the average as the result. The details are in Table 2. Note that protocol Γ_{sag} needs around 22.81 seconds in our setting to negotiate keys between clients, and we do not include it in Table 2.

Table 2. Offline performances(seconds, 100 epochs).

Protocol / Model	Γ_{fl}	Γ_{sag}	Γ_{sash}	Γ_{sam}
Linear Regression	0	11.90	0.60	1.19
MLP	0	110.53	14.24	27.47
CNN	0	243.92	21.77	41.44

From Table 2 we can see that Γ_{sash} is around 10× faster than Γ_{sag}, and Γ_{sam} is around 5× faster than Γ_{sag}.

As for the online phase, we plot the running time along with the epochs for CNN in Fig 5(b). It is illustrated that our semi-honest protocol Γ_{sash} is faster than Γ_{sag} even Γ_{sag} is in the best situation. For example, the running time for Γ_{sash} is 701.85 seconds, while Γ_{sag} needs 744.68 seconds, both run 100 epochs. As for our malicious protocol Γ_{sam}, the running time is a little longer, which is around 819.99 seconds.

However, we know that Γ_{sag} needs extra time to deal with the masking elements if some clients dropped out in the training process, this reduces their efficiency greatly, while our protocols do not need this extra operation even if the same misfortune happens. This is because the clients in our protocols are independent of each other, which means the dropped clients do not impact other online clients. Combining offline and online, we can see that our protocol's whole performance is better than Γ_{sag}, both semi-honest and malicious. And compared to privacy-preserving deep learning methods based on HE[28], our improvements are dramatically where they need 2.25 hours for MLP and 7.3 hours for CNN. Note that we plot the figures for linear regression and MLP in **Appendix B**.

On the other hand, $\mathit{\Gamma}_{sag}$ only needs one parameter server, but $\mathit{\Gamma}_{sash}$ and $\mathit{\Gamma}_{sam}$ need two or more parameter servers, and we require that at most $t-1$ servers could collude. However, import two or more servers is a common method for distributing machine learning, and it is easy to satisfy this security requirement in practice. Meanwhile, in $\mathit{\Gamma}_{sag}$ the parameter server would learn the aggregated gradients and even the trained model, but in $\mathit{\Gamma}_{sash}$ and $\mathit{\Gamma}_{sam}$ the parameter servers can learn nothing. What's more, our protocols are more robust than $\mathit{\Gamma}_{sag}$, since we allow $n-t$ servers to halt at worst but $\mathit{\Gamma}_{sag}$ must ensure the parameter server runs normally.

5 Conclusion

We introduce a novel secure computation framework for distributed machine learning that achieves high accuracy and performance by combining distributed machine learning with secret sharing. In contrast to previous state-of-the-art frameworks, we improve the efficiency of more than $10\times$ and maintain the required security in semi-honest. Besides, we propose a verifiable protocol against servers-only malicious modifications based on information-theoretic MAC. We evaluated our framework on linear regression, MLP, and CNN and achieve excellent results both in accuracy and performance. What's more, combining differential privacy with distributed machine learning is a promising solution against inferring attacks, which can enhance our security as well. We leave it for future work.

6 Acknowledgements

We are grateful to the anonymous reviewers for their comprehensive comments. And we thank Xiangfu Song, Yiran Liu from Shandong University for helpful discusssions on MPC, and Junming Ke from Singapore University of Technology and Design for his help. This work was supported by

References

1. Shamir A. How to share a secret[J]. Communications of the ACM, 1979, 22(11): 612-613. https://doi.org/10.1145/359168.359176
2. Lindell Y, Pinkas B. Privacy-preserving data mining[C]//Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2000: 36-54. https://doi.org/10.1145/335191.335438
3. Du W, Atallah M J. Privacy-preserving cooperative scientific computations[C]//csfw. IEEE, 2001: 0273. https://doi.org/10.1109/CSFW.2001.930152

4. Du W, Han Y S, Chen S. Privacy-preserving multivariate statistical analysis: Linear regression and classification[C]//Proceedings of the 2004 SIAM international conference on data mining. Society for Industrial and Applied Mathematics, 2004: 222-233. https://doi.org/10.1137/1.9781611972740.21
5. Sanil A P, Karr A F, Lin X, et al. Privacy-preserving regression modelling via distributed computation[C]//Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2004: 677-682. https://doi.org/10.1145/1014052.1014139
6. Jagannathan G, Wright R N. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data[C]//Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. ACM, 2005: 593-599. https://doi.org/10.1145/1081870.1081942
7. Yu H, Vaidya J, Jiang X. Privacy-preserving svm classification on vertically partitioned data[C]//Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, Berlin, Heidelberg, 2006: 647-656. https://doi.org/10.1007/11731139$_7$4
8. Slavkovic A B, Nardi Y, Tibbits M M. " Secure" Logistic Regression of Horizontally and Vertically Partitioned Distributed Databases[C]//Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007). IEEE, 2007: 723-728. https://doi.org/10.1109/ICDMW.2007.114
9. Bunn P, Ostrovsky R. Secure two-party k-means clustering[C]//Proceedings of the 14th ACM conference on Computer and communications security. ACM, 2007: 486-497. https://doi.org/10.1145/1315245.1315306
10. Vaidya J, Yu H, Jiang X. Privacy-preserving SVM classification[J]. Knowledge and Information Systems, 2008, 14(2): 161-178. https://doi.org/10.1007/s10115-007-0073-7
11. Bottou L. Large-scale machine learning with stochastic gradient descent[M]//Proceedings of COMPSTAT'2010. Physica-Verlag HD, 2010: 177-186. https://doi.org/10.1007/978-3-7908-2604-3$_1$6
12. Damgrd I, Pastro V, Smart N, et al. Multiparty computation from somewhat homomorphic encryption[C]//Annual Cryptology Conference. Springer, Berlin, Heidelberg, 2012: 643-662. https://doi.org/10.1007/978-3-642-32009-5$_3$8
13. Nikolaenko V, Ioannidis S, Weinsberg U, et al. Privacy-preserving matrix factorization[C]//Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. ACM, 2013: 801-812. https://doi.org/10.1145/2508859.2516751
14. Wu S, Teruya T, Kawamoto J. Privacy-preservation for stochastic gradient descent application to secure logistic regression[J]. The 27th Annual Conference of the Japanese Society for Artificial Intelligence 27, 2013, 1-4.
15. Song S, Chaudhuri K, Sarwate A D. Stochastic gradient descent with differentially private updates[C]//2013 IEEE Global Conference on Signal and Information Processing. IEEE, 2013: 245-248. https://doi.org/10.1109/GlobalSIP.2013.6736861
16. Li M, Andersen D G, Park J W, et al. Scaling distributed machine learning with the parameter server[C]//11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014: 583-598. https://doi.org/10.1145/2640087.2644155
17. Shokri R, Shmatikov V. Privacy-preserving deep learning[C]//Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. ACM, 2015: 1310-1321. https://doi.org/10.1145/2810103.2813687
18. Konen J, McMahan H B, Yu F X, et al. Federated learning: Strategies for improving communication efficiency[J]. arXiv preprint arXiv:1610.05492, 2016.

19. Abadi M, Chu A, Goodfellow I, et al. Deep learning with differential privacy[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016: 308-318. https://doi.org/10.1145/2976749.2978318
20. Gascn A, Schoppmann P, Balle B, et al. Secure Linear Regression on Vertically Partitioned Datasets[J]. IACR Cryptology ePrint Archive, 2016, 2016: 892.
21. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (GDPR). Official Journal of the European Union, L119, May 2016.
22. Gilad-Bachrach R, Dowlin N, Laine K, et al. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy[C]//International Conference on Machine Learning. 2016: 201-210.
23. Mohassel P, Zhang Y. Secureml: A system for scalable privacy-preserving machine learning[C]//2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017: 19-38. https://doi.org/10.1109/SP.2017.12
24. Liu J, Juuti M, Lu Y, et al. Oblivious neural network predictions via minionn transformations[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017: 619-631. https://doi.org/10.1145/3133956.3134056
25. Bonawitz K, Ivanov V, Kreuter B, et al. Practical secure aggregation for privacy-preserving machine learning[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017: 1175-1191. https://doi.org/10.1145/3133956.3133982
26. Lin Y, Han S, Mao H, et al. Deep gradient compression: Reducing the communication bandwidth for distributed training[J]. arXiv preprint arXiv:1712.01887, 2017.
27. Riazi M S, Weinert C, Tkachenko O, et al. Chameleon: A hybrid secure computation framework for machine learning applications[C]//Proceedings of the 2018 on Asia Conference on Computer and Communications Security. ACM, 2018: 707-721. https://doi.org/10.1145/3196494.3196522
28. Phong L T, Aono Y, Hayashi T, et al. Privacy-preserving deep learning via additively homomorphic encryption[J]. IEEE Transactions on Information Forensics and Security, 2018, 13(5): 1333-1345. https://doi.org/10.1109/TIFS.2017.2787987
29. Wagh S, Gupta D, Chandran N. SecureNN: 3-Party Secure Computation for Neural Network Training[J]. Proceedings on Privacy Enhancing Technologies, 2019, 1: 24. https://doi.org/10.2478/popets-2019-0035
30. Nasr M, Shokri R, Houmansadr A. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks[J]. arXiv preprint arXiv:1812.00910, 2018.
31. Yang Q, Liu Y, Chen T, et al. Federated machine learning: Concept and applications[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2019, 10(2): 12. https://doi.org/10.1145/3298981
32. Juvekar C, Vaikuntanathan V, Chandrakasan A. GAZELLE: A Low Latency Framework for Secure Neural Network Inference[C]//27th USENIX Security Symposium (USENIX Security 18). 2018: 1651-1669.
33. Centers for Medicare & Medicaid Services. The Health Insurance Portability and Accountability Act of 1996 (HIPAA). Online at http://www.cms.hhs.gov/hipaa/, 1996

A Proof of Correctness

A.1 Lemma 1

Proof. Suppose we have two secrets, s_0 and s_1, and we share both in Shamir's Secret Sharing scheme with two polynomial-functions

$$\begin{aligned} f(x) &= a_0 + a_1 \cdot x + ... + a_{t-1} \cdot x^{t-1} \mod p \\ g(x) &= b_0 + b_1 \cdot x + ... + b_{t-1} \cdot x^{t-1} \mod p \end{aligned} \tag{12}$$

where $f(0) = a_0 = s_0$, $g(0) = b_0 = s_1$ and p is a large prime.

In order to compute the shares, we can evaluate $f(x)$ and $g(x)$ at n different points $f(x_0), f(x_1), ..., f(x_{n-1})$ and $g(x_0), g(x_1), ..., g(x_{n-1})$ respectively.

Then we will turn to getting the shares of s_0+s_1. We define a new polynomial-function

$$h(x) = (a_0 + b_0) + (a_1 + b_1) \cdot x + ... + (a_{t-1} + b_{t-1}) \cdot x^{t-1} \mod p \tag{13}$$

Obviously, $h(x)$ is a polynomial-function of degree $t-1$ with t cofficients and $h(0) = s_0 + s_1$. On the one hand, $h(x_i)$ is the shares for $s_0 + s_1$, and on the other hand, we can confirm

$$\begin{aligned} h(x_i) &= (a_0 + b_0) + (a_1 + b_1) \cdot x_i + ... + (a_{t-1} + b_{t-1}) \cdot x_i^{t-1} \\ &= (a_0 + a_1 \cdot x_i + ... + a_{t-1} \cdot x_i^{t-1}) + (b_0 + b_1 \cdot x_i + ... + b_{t-1} \cdot x_i^{t-1}) \\ &= f(x_i) + g(x_i) \mod p, \ 0 \le i \le n-1 \end{aligned} \tag{14}$$

So that the shares of $s_0 + s_1$ can be computed by adding the corresponding shares of s_0 and s_1.

A.2 Lemma2

Proof. Suppose we have $x_0, x_1, ..., x_{n-1}$ and a secret key α. We could the compute the MAC of x_i

$$\boldsymbol{\delta}(x_i) = \alpha \cdot x_i \mod p, \ 0 \le i \le n-1 \tag{15}$$

Then we can also compute the MAC of x_i's sum:

$$\boldsymbol{\delta}(\sum_{i=0}^{n-1} x_i) = \alpha \cdot (\sum_{i=0}^{n-1} x_i) \mod p \tag{16}$$

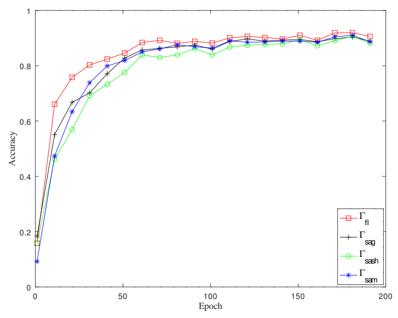
Then it is easy to confirm

$$\begin{aligned} \boldsymbol{\delta}(\sum_{i=0}^{n-1} x_i) &= (\alpha \cdot x_0) + (\alpha \cdot x_1) + ... + (\alpha \cdot x_{n-1}) \mod p \\ &= \boldsymbol{\delta}(x_0) + \boldsymbol{\delta}(x_1) + ... + \boldsymbol{\delta}(x_{n-1}) \mod p \\ &= \sum_{i=0}^{n-1} \boldsymbol{\delta}(x_i) \mod p \end{aligned} \tag{17}$$

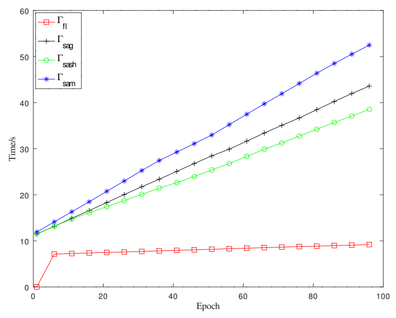
For a more concrete proof, please refer to[12].

B Accuracy and Performance for linear regression and MLP

B.1 Linear regression



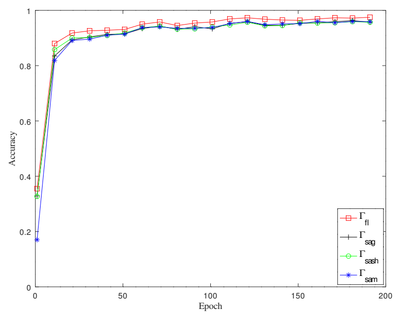
(a)



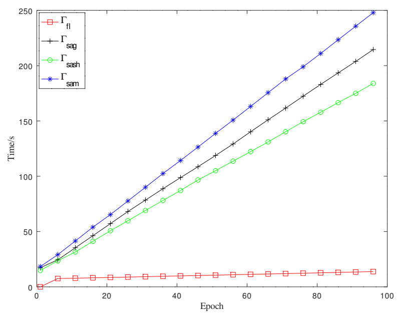
(b)

Fig. 6. Experimental results of linear regression. (a) is for accuracy, (b) is for performance.

B.2 MLP



(a)



(b)

Fig. 7. Experimental results of MLP. (a) is for accuracy, (b) is for performance.