

PrivColl: Practical Privacy-Preserving Collaborative Machine Learning

Yanjun Zhang, Guangdong Bai (✉), Xue Li, Caitlin Curtis,
Chen Chen, and Ryan K L Ko

The University of Queensland, St Lucia, Queensland, Australia
{yanjun.zhang, g.bai, c.curtis, chen.chen, ryan.ko}@uq.edu.au
xueli@itee.uq.edu.au

Abstract. Collaborative learning enables two or more participants, each with their own training dataset, to collaboratively learn a joint model. It is desirable that the collaboration should not cause the disclosure of either the raw datasets of each individual owner or the local model parameters trained on them. This privacy-preservation requirement has been approached through differential privacy mechanisms, homomorphic encryption (HE) and secure multiparty computation (MPC), but existing attempts may either introduce the loss of model accuracy or imply significant computational and/or communicational overhead.

In this work, we address this problem with the lightweight additive secret sharing technique. We propose PRIVCOLL, a framework for protecting local data and local models while ensuring the correctness of training processes. PRIVCOLL employs secret sharing technique for securely evaluating addition operations in a multiparty computation environment, and achieves practicability by employing only the homomorphic addition operations. We formally prove that it guarantees privacy preservation even though the majority ($n - 2$ out of n) of participants are corrupted. With experiments on real-world datasets, we further demonstrate that PRIVCOLL retains high efficiency. It achieves a speedup of more than 45X over the state-of-the-art MPC-/HE-based schemes for training linear/logistic regression, and 216X faster for training neural network.

Keywords: privacy · machine learning · collaborative learning.

1 Introduction

The performance of machine learning largely relies on the availability of datasets. To take advantage of massive data owned by multiple entities, collaborative machine learning has been proposed to enable two or more data owners to construct a joint model. One typical scenario demanding collaborative learning is where the *features* of a same sample are held by multiple data owners. The collaboration among owners can improve the model accuracy by leveraging additional features from each other. A real-world example is that a recommender system can take use of the ratings of a same item among multiple online merchants to enhance its predictive power.

To address the privacy concerns arising from collaborative learning, many studies [7, 12, 40, 44] have been proposed to provide *data locality* by distributing learning algorithms onto data owners such that the data can be confined within their owners. Despite this, their learning processes still entail sharing locally trained models, in order to synthesize the final models. However these local models are subject to information leakage. For example, model-inversion attacks [10, 30] are able to restore training data from them. In addition, in the scenarios where the model itself represents intellectual property, e.g., in financial market systems, it is an essential requirement for the local models to be kept confidential [32].

To provide a supplementary, i.e., *privacy-preserving synthesis of the local models*, differential privacy mechanisms and cryptographic mechanisms may be employed. The former [1, 8, 19, 23, 37, 38, 48] usually entails adding noise on the model parameters, causing loss in the accuracy of the final models. The cryptographic mechanisms such as homomorphic encryption (HE) [7, 29, 36] and secure multiparty computation (MPC) [5, 11, 12, 28, 31] are able to yield identical models as those trained on plaintext data, but are known to be limited by the significant computational or communicational overheads.

This work focuses on the practicability of the cryptographic solutions. We propose a lightweight framework named PRIVCOLL for privacy-preserving collaborative learning in the distributed feature scenario. PRIVCOLL adopts the two-layer architecture commonly used in previous privacy-preserving collaborative learning frameworks [23, 31, 40]. It has a local node layer consisting of participating data owners, and an aggregation node (which can be untrustworthy). The main strategy of PRIVCOLL is to dispense the homomorphic *multiplication operations* and *non-linear functions* on ciphertext, as they are far more costly in computation and communication than the *addition operations* [3, 13, 15, 42]. To this end, we redesign the workflow of collaborative learning, so that it employs only the homomorphic addition operations provided by *additive secret sharing scheme* [4] for synthesizing local models and intermediate outcomes. The computation that is carried out by the aggregation mode uses only the *sum* of the intermediate results that are generated by the local nodes (detailed in Section 3.3). As such, PRIVCOLL achieves significant cost savings, in comparison with state-of-the-art cryptographic solutions.

The redesigned workflow also ensures that both the raw data and local models are always kept with their owners. We formally prove PRIVCOLL preserves privacy in such a way that the *honest-but-curious* participants, who have access to the sum of the intermediate outcomes produced by the local nodes and additional knowledge learned from the training iterations, are unlikely (i.e., with a negligible probability ε) to reveal the raw training data or the local model parameters of other participants.

Notably, our new collaborative learning workflow in PRIVCOLL introduces no sacrifice to the accuracy of the models, and also supports a wide range of machine learning algorithms as previous work does [1, 23, 31]. Intuitively, our solution makes use of the chain rules in calculus to decompose gradient descent

optimization into computational primitives, and to distribute them to the local nodes and the aggregation node respectively. When they collaborate together, these primitives can be recombined to achieve the correctness of learning. We prove that such correctness is guaranteed for any algorithm that uses gradient descent for optimization, including but not limited to linear regression, logistic regression, and a variety of neural networks.

Contributions. In general, our contributions can be summarized as follows.

- **A Novel Privacy-preserving Collaborative Framework.** We propose a novel framework PRIVCOLL for collaborative learning with distributed features. It preserves privacy while enabling a wide range of machine learning algorithms and achieving high computation efficiency. Not only does PRIVCOLL achieve the data locality as previous work does, but it also keeps the local models confidential.
- **Provable Privacy Preservation and Correctness Guarantee.** We prove the privacy preservation of PRIVCOLL, demonstrating a negligible probability of corrupted parties revealing either the original data or the trained parameters from other honest parties. We also prove that PRIVCOLL ensures the learned model is identical to that in the traditional non-distributed framework.
- **Experimental Evaluations.** We conduct experiments on real datasets, showing that PRIVCOLL achieves a significant improvement of efficiency over the state-of-the-art cryptographic solutions based on MPC and HE. For example, PRIVCOLL achieves around 22.5 minutes for a two-hidden-layer neural network to process all samples in the MNIST dataset [27], while it takes more than 81 hours with a state-of-the-art MPC protocol SecureML [31].

2 Background

In this section, we introduce the background knowledge that is necessary to understand our framework.

2.1 Gradient Descent Optimization

Gradient descent is by far the most commonly used optimization strategy among various machine learning and deep learning algorithms. It is used to find the values of coefficients that minimize a cost function as far as possible. Given a defined cost function J , the coefficient matrix W is derived by the optimization $\arg \min_W J$, and is updated as:

$$W := W - \alpha \frac{\partial J}{\partial W} \quad (1)$$

Given a particular training dataset X and a label matrix y , the cost function J can be defined as $J(\sigma(XW), y, W)$, where σ is determined by the learning model. For example, in logistic regression, σ is usually a sigmoid function $1/(1 +$

e^{-z}), while in neural network, σ is a composite function that is known as forward propagation. According to the chain rule in calculus, the gradient with respect to W is computed as $\frac{\partial J}{\partial \sigma} \frac{\partial \sigma}{\partial XW} \frac{\partial XW}{\partial W} + \tau$, where τ is the gradient with respect to the regularization term, which is independent to X . Let Δ be $\frac{\partial J}{\partial \sigma} \frac{\partial \sigma}{\partial XW}$, and $\frac{\partial XW}{\partial W}$ equals to X , then the gradient with respect to W can be written as:

$$\frac{\partial J}{\partial W} = \Delta X + \tau. \quad (2)$$

As such, we can decompose the gradient descent optimization into Δ , X and τ , in which Δ is a function of XW . This provides an algorithmic foundation for PRIVCOLL's distribution of learning algorithms.

2.2 Additive Secret Sharing Scheme

Secret sharing schemes aim to securely distribute secret values amongst a group of participants. PRIVCOLL employs the secret sharing scheme proposed by [4], which uses *additive sharing* over $\mathbb{Z}_{2^{32}}$. In this scheme, a secret value srt is split to s shares $E_{srt}^1, \dots, E_{srt}^s \in \mathbb{Z}_{2^{32}}$ such that

$$E_{srt}^1 + E_{srt}^2 + \dots + E_{srt}^s \equiv srt \pmod{2^{32}}, \quad (3)$$

and any $s - 1$ elements $E_{srt}^{i_1}, \dots, E_{srt}^{i_{s-1}}$ are uniformly distributed. This prevents any participant who has part of the shares from deriving the value of srt , unless all participants join their shares.

In addition, the scheme has a homomorphic property that allows efficient and secure addition on a set of secret values srt_1, \dots, srt_s held by corresponding participants S_1, \dots, S_s . To do this, each participant S_i executes a randomised sharing algorithm $Shr(srt_i, S)$ to split its secret srt_i into shares $E_{srt_i}^1, \dots, E_{srt_i}^s$, and distributes each $E_{srt_i}^j$ to the participant S_j . Then, each S_i locally adds the shares it holds, $E_{srt_1}^i, \dots, E_{srt_s}^i$, to produce $\sum_{j=1}^s E_{srt_j}^i$ (denoted by E^i for brevity). After that, a reconstruction algorithm $Rec(\{(E^i, S_i)\}_{S_i \in S})$, which takes E^i from each participant and add them together, can be executed by an aggregator to reconstruct the $\sum_{i=1}^s srt_i$ without revealing any secret addends srt_i .

3 Design of PrivColl

3.1 Scope and Threat Model

The involved parties in PRIVCOLL are a set of local nodes (i.e., data owners) S_1, \dots, S_s and an aggregation node Agg . Each local node holds part of features of the training samples, denoted by X^l ($l \in \{1, \dots, s\}$), and the corresponding local model W^l ($l \in \{1, \dots, s\}$) trained on X^l . Each $X^l \in \mathbb{R}^{m \times d_l}$ is a $m \times d_l$ matrix representing m training samples with d_l features, and $W^l \in \mathbb{R}^{d_l \times k}$ is a matrix of coefficients, where k is the number of output classes. In PRIVCOLL, m and k are public and known by every party, and d_l is private and only known by the

corresponding data owner S_l . We use X to denote the vertical concatenation of the local training datasets X^1, \dots, X^s . Then we know that X is a $m \times n$ matrix where $n = \sum_{l=1}^s d_l$ (i.e., the total number of features in X). Since d_l is private, n is unknown unless all of the local nodes join their views.

PRIVCOLL aims to defend against an honest-but-curious adversary \mathcal{A} , who follows the collaboration protocols and training procedures, but is intending to obtain the datasets of other local nodes (i.e., X^l) and/or the model parameters trained out of them (i.e., W^l). The adversary may control Agg , and t out of s local nodes. Here, we conservatively assume $t < s - 1$, which implies that at least two local nodes need to be out of the adversary's control, as $s - 1$ comprised local nodes who has the sum of their shares, colluding with Agg who has the sum of all shares, will be able to obtain the share of the remaining node by a simple subtraction (detailed in Section 4).

3.2 Definitions of Privacy Preservation and Correctness

Keeping local data/model private and providing functional correctness are the main properties PRIVCOLL aims to achieve. Below we present the definitions of these two properties.

Definition 1. (ε -privacy) *A mechanism preserves ε -privacy if the probability for a probabilistic polynomial-time (PPT) adversary to derive X^l or W^l of any benign node S_l based on its knowledge is not greater than ε .*

Definition 2. (Correctness) *Given a function \mathcal{F} that takes as input a training dataset X , and its distributed version \mathcal{F}' that takes as inputs X 's vertical partitions X^1, \dots, X^s , we say \mathcal{F}' is correct if $\mathcal{F}'(\{(S_l, X^l), Agg\}_{S_l \in S}) = \mathcal{F}(X)$.*

3.3 Workflow of PrivColl

Figure 1 illustrates the end-to-end workflow of PRIVCOLL, which is divided into the following steps.

Initialization: Each local node S_l holds its own training dataset $X^l \in \mathbb{R}^{m \times d_l}$, and randomly initializes its coefficient matrix $W^l \in \mathbb{R}^{d_l \times k}$.

Step 1: In each iteration of gradient descent, each S_l multiplies X^l by W^l locally, resulting in a $X^l W^l \in \mathbb{R}^{m \times k}$. The value of d_l , i.e., the number of features in X^l , is removed by such a matrix multiplication.

Step 2: Each S_l executes the sharing algorithm Shr to split $X^l W^l$ into s shares using the *additive secret sharing scheme*.

$$\{(S_i, E_{X^l W^l}^i)\}_{S_i \in S} \leftarrow Shr(X^l W^l, S), \quad (4)$$

in which Shr takes as input a secret $X^l W^l$ and a set S of local nodes, and produces a set of shares $E_{X^l W^l}^i (i \in \{1, \dots, s\})$, each of which is distributed to a different $S_i \in S$. Then, each S_l calculates the sum of all shares it receives, and gets $E^l = \sum_{j=1}^s E_{X^j W^j}^l$.

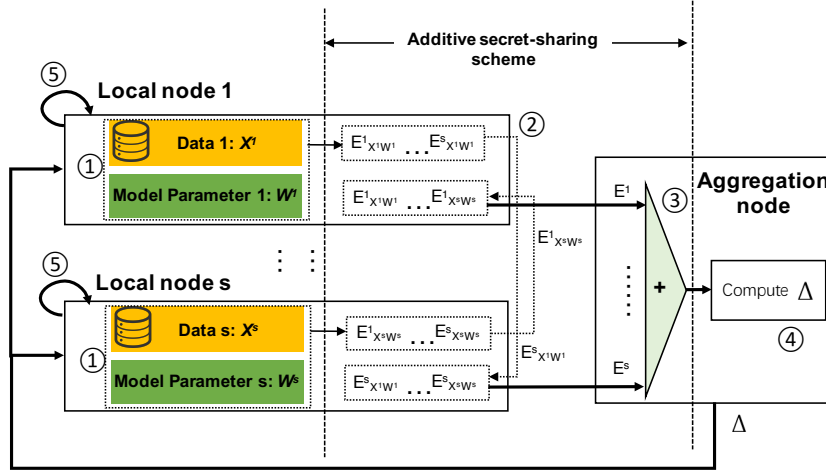


Fig. 1: The end-to-end workflow of the PRIVCOLL

Step 3: *Agg* collects all E^l from local nodes and add them together. Since

$$\sum_{l=1}^s E^l = \sum_{l=1}^s \sum_{j=1}^s E_{X^j W^j}^l = \sum_{j=1}^s \sum_{l=1}^s E_{X^j W^j}^l = \sum_{j=1}^s X^j W^j, \quad (5)$$

this addition reconstructs the homomorphic addition result XW (equals $\sum_{l=1}^s X^l W^l$).

Step 4: *Agg* computes $\Delta = \frac{\partial J}{\partial \sigma} \frac{\partial \sigma}{\partial XW}$ (c.f., Equation 2), and sends Δ back to the local nodes.

Step 5: With the received Δ , each S_l updates its local coefficient matrix W^l .

$$W^l \leftarrow W^l - \alpha(\Delta X^l + \tau^l) \quad (6)$$

Step 1-5 are repeated for the next training iteration until convergence.

4 Privacy Preservation Analysis

In this section, we analyze the privacy preservation of PRIVCOLL's learning process. To this end, we first investigate the overall knowledge that can be learned by an adversary from the training iterations (Section 4.2). Then we prove that the knowledge is limited such that the desired ε -privacy (Definition 1) is achieved with a negligible ε (Section 4.3).

4.1 Preliminaries

We start with the following lemma that is soon used in our proof.

Lemma 1. Consider a positive (semi-)definite matrix A that is obtained as the product of a real number matrix B by its transpose B^T

$$A = BB^T, \quad (7)$$

where B is of rank r . Without knowing the number of columns of B , the probability of solving the B given A , denoted as $P(B|A)$, is $\leq (r!)^{-1}$.

Proof. Since the matrix A is positive (semi-)definite, there exists an eigen-decomposition such that

$$A = U\Lambda U^T, \quad (8)$$

where U denotes a matrix of eigenvectors of A (each column of U is an eigenvector of A), and Λ denotes a diagonal matrix whose diagonal elements are the eigenvalues. For a positive (semi-)definite matrix, all the eigenvalues are non-negative. A different ordering of the eigenvector columns results in a different U and a corresponding Λ [16].

With the eigen-decomposition, B can be constructed by $B = U\Lambda^{\frac{1}{2}}$, where $\Lambda^{\frac{1}{2}}$ has the square roots of eigenvalues as its diagonal elements, and all its remaining values are zeros. Each eigen-decomposition leads to a different U and a corresponding Λ , resulting in a unique solution of B . A matrix B of rank r has r non-zero eigenvalues and thus there are $r!$ different possible orderings of eigenvector columns, implying $r!$ different U s and the corresponding Λ s. Consequently, there are $r!$ different possible solutions of computing B , which gives the probability of solving B given A with eigen-decomposition $P_{eigen}(B|A) = (r!)^{-1}$.

In addition, without the knowledge of the number of columns in B , there are more than $r!$ possible solutions of solving B . This is because from the eigen-decomposition construction, B 's columns are orthogonal. In general, the matrix B need not have orthogonal columns (it can be rectangular) [22]. Thus, $P(B|A) \leq (r!)^{-1}$.

4.2 Party Knowledge

We define the *party knowledge* as the overall knowledge that can be learned by adversary parties. It includes the parties' own inputs, and the additional knowledge that can be inferred from the training iterations. We prove that the party knowledge in PRIVCOLL is bounded within a certain range. In particular, we demonstrate that the overall *party knowledge* of adversary party I in PRIVCOLL is a set of $\{X^{I'} \mid I' \in I, W^{I'} \mid I' \in I, XW, \sum X^l W^l \mid l \in S \setminus I, \sum X^l (X^l)^T \mid l \in S \setminus I, XX^T\}$, where $\{X^{I'} \mid I' \in I, W^{I'} \mid I' \in I, XW\}$ are the adversary's own input in the workflow, and $\{\sum X^l W^l \mid l \in S \setminus I, \sum X^l (X^l)^T \mid l \in S \setminus I, XX^T\}$ are the additional information that can be inferred from the training iterations. The party knowledge we derive in this Section will be used in Section 4.3 to prove the privacy preservation of PRIVCOLL.

We use the simulation paradigm (also known as the *real/ideal model*) [15] to prove such a bound of party knowledge. The simulation paradigm compares what an adversary can do in a real protocol execution *REAL* to what it can do

in an ideal setting with a trusted functionality (simulation) SIM [15]. Formally, the protocol \mathcal{P} securely computes a functionality $\mathcal{F}_{\mathcal{P}}$ if for every adversary in $REAL$, there exists an adversary in SIM , such that the view of the adversary from $REAL$ is indistinguishable from the view of the adversary from SIM . A perfect indistinguishability [6] between the view of $REAL$ and SIM guarantees that the adversary, without error probability, can learn nothing more than their own inputs and the information required by SIM for the simulation.

We introduce some notations used in our proof. We use $X^{S'} = \{X^l | S_l \in S'\}$ to indicate the inputs of any subset of local nodes $S' \subseteq S$. Given any subset $I_0 \subseteq S$ of the parties *without* the knowledge of XW , and subset $I_1 \subseteq S \cup \{Agg\}$ of the parties *with* the knowledge of XW , let $REAL(X^S, XW, S, t, \mathcal{P}, I)$ denote the combined views of all parties in $I = I_0 \cup I_1$ from the execution of a real protocol \mathcal{P} , where t is the adversary threshold (recall that $t < s - 1$). Let $SIM(X^I, Z, S, t, \mathcal{F}_{\mathcal{P}}, I)$ denote the views of I from an ideal execution, where Z is the information required by SIM for the simulation. In other words, the Z indicates the party knowledge that the adversary can and only can learn other than their own inputs.

Theorem 1. (Party Knowledge) *The simulator $SIM(X^I, Z, S, t, \mathcal{F}_{\mathcal{P}}, I)$ is perfectly indistinguishable from $REAL$ with respect to their outputs, namely*

$$REAL(X^S, XW, S, t, \mathcal{P}, I) \equiv SIM(X^I, Z, S, t, \mathcal{F}_{\mathcal{P}}, I)$$

if and only if

$$Z = \{z_1 = \sum X^l W^l \mid_{l \in S \setminus I}, z_2 = \sum X^l (X^l)^T \mid_{l \in S \setminus I}, z_3 = X X^T\}.$$

Proof. We define the simulator through each of the i^{th} training iteration as:

- SIM_0 : This is the simulator for the **Initialization**. In the step of initialization, the view of parties in $I_0 \cup I_1$ does not depend on the inputs of the parties not in $I_0 \cup I_1$. Therefore, instead of sending the actual $\{X^l W^{l(0)}\}_{l \in S \setminus \{I_0 \cup I_1\}}$ of the parties $S \setminus \{I_0 \cup I_1\}$ to the aggregation node, the simulator can produce a simulation by running the parties $S \setminus \{I_0 \cup I_1\}$ on a pseudorandom vector $\mu^{l(0)}$ in \mathbb{R}^m as input, and then output the same pseudorandom vector to the aggregation node. Since the model parameter W^l is also randomized in the step of initialization, the pseudorandom vectors for the inputs of all honest parties $S \setminus \{I_0 \cup I_1\}$, and the joint view of parties in $\{I_0 \cup I_1\}$ will be identical to that in $REAL$

$$\{\mu^{l(0)} \mid_{l \in S \setminus \{I_0 \cup I_1\}}, X^{l'} W^{l'(0)} \mid_{l' \in \{I_0 \cup I_1\}}\} \equiv \{X^S W^{S(0)}\}.$$

- $SIM_i, i \geq 1$: This is the simulator for the i^{th} training iteration ($i \geq 1$). The simulator computes $\Delta^{(i)} = f(\Sigma^{(i)})$, where the function f is determined by the learning model. For example, in the linear regression, $f(x) = x$, while in logistic regression, $f(x) = 1/(1 + e^{-x})$.

We respectively consider the simulator for I_0, I_1 . First, with respect to I_0 , the simulator computes $\Sigma^{(i)}$ as

$$\Sigma^{(i)} = \sum \mu^{l(i)} |_{l \in S \setminus I_0} + \sum X^{l'} W^{l'(i)} |_{l' \in I_0} - y,$$

where $\mu^{l(i)}$ is computed by the result from the previous iteration as

$$\mu^{l(i)} = z_1^{l(i-1)} - \frac{\alpha}{m} z_2^l \Delta^{(i-1)} |_{l \in S \setminus I_0}.$$

Therefore,

$$\sum \mu^{l(i)} |_{l \in S \setminus I_0} = \sum z_1^{l(i-1)} - \frac{\alpha}{m} \sum z_2^l \Delta^{(i-1)} |_{l \in S \setminus I_0}.$$

Note $X^{l'} W^{l'(i)}$ is also computed by the result from the previous iteration, and

$$\sum X^{l'} W^{l'(i)} |_{l' \in I_0} = \sum X^{l'} W^{l'(i-1)} - \frac{\alpha}{m} \sum X^{l'} (X^{l'})^T \Delta^{(i-1)} |_{l' \in I_0}.$$

Then, the $\Sigma^{(i)}$ can be written as

$$\begin{aligned} \Sigma^{(i)} &= \sum z_1^{l(i-1)} |_{l \in S \setminus I_0} + \sum X^{l'} W^{l'(i-1)} |_{l' \in I_0} \\ &\quad - \frac{\alpha}{m} (\sum z_2^l \Delta^{(i-1)} |_{l \in S \setminus I_0} + \sum X^{l'} (X^{l'})^T \Delta^{(i-1)} |_{l' \in I_0}) - y. \end{aligned}$$

Note that, in *REAL*, the output of $\Sigma^{(i)}$ by S is

$$\sum X^S W^{S(i)} - y = \sum X^S W^{S(i-1)} - \frac{\alpha}{m} \sum X^s (X^s)^T \Delta^{(i-1)} - y.$$

Thus, $\{\sum z_1^{l(i-1)} |_{l \in S \setminus I_0}, \sum z_2^l |_{l \in S \setminus I_0}, \sum X^{l'} W^{l'(i-1)} |_{l' \in I_0}, \sum X^{l'} (X^{l'})^T |_{l' \in I_0}\}$ which is the joint view of all honest parties $S \setminus I_0$ and parties in I_0 will be perfectly indistinguishable to $\{\sum X^S W^{S(i-1)}, \sum X^s (X^s)^T\}$ which is the output in *REAL*.

Next, we consider the simulator for I_1 . With respect to I_1 , we let the simulator compute $\Sigma^{(i)}$ as

$$\Sigma^{(i)} = \mu^{(i)} - y,$$

where $\mu^{(i)}$ is computed by the result from the previous iteration as

$$\mu^{(i)} = XW^{(i-1)} - \frac{\alpha}{m} z_3 \Delta^{(i-1)}.$$

Then, the $\Sigma^{(i)}$ can be written as

$$\Sigma^{(i)} = XW^{(i-1)} - \frac{\alpha}{m} z_3 \Delta^{(i-1)} - y,$$

Note that, in *REAL*, the output of $\Sigma^{(i)}$ by S is

$$XW^{(i)} - y = XW^{(i-1)} - \frac{\alpha}{m} XX^T \Delta^{(i-1)} - y.$$

Thus, the joint view of all parties in *SIM* with knowledge of z_3 will be perfectly indistinguishable to $\{XW^{(i-1)}, XX^T\}$ which is the output in *REAL*.

All in all, the output of the simulator SIM of each training iteration is perfectly indistinguishable from the output of $REAL$. For the simulator with respect to $I = I_0 \cup I_1$, knowledge of $z_1 = \sum z_1^{l(i-1)} |_{l \in S \setminus I} = \sum X^l W^l |_{l \in S \setminus I}$, $z_2 = \sum z_2^l |_{l \in S \setminus I} = \sum X^l (X^l)^T |_{l \in S \setminus I}$ and $z_3 = XX^T$ is sufficient, completing the proof.

4.3 Privacy Preservation Guarantee

With lemma introduced in Section 4.1, and party knowledge discussed in Section 4.2, we give our theorem of privacy preservation guarantee.

Theorem 2. *Let I denote the adversary party with party knowledge $\{X^{l'} |_{l' \in I}, W^{l'} |_{l' \in I}, XW, \sum X^l W^l |_{l \in S \setminus I}, \sum X^l (X^l)^T |_{l \in S \setminus I}, XX^T\}$. Let X^H denote the vertical concatenation of $\{X^l |_{l \in S \setminus I}\}$, which is the concatenation of honest local nodes' training datasets. Let r denote the rank of X^H . PRIVCOLL preserves ε -privacy against adversary party I on the training dataset $X \in \mathbb{R}^{m \times n}$, where $\varepsilon \leq (r!)^{-1}$.*

Proof. We give some sketches here.

We start with the party knowledge XX^T . In PRIVCOLL , n (i.e. the number of column of X) is unknown given the adversary threshold $t < s - 1$. In addition, the rank of X , denoted as R , is $> r$. Invoking Lemma 1 gives that the probability of solving the X of rank R given XX^T is $\leq (R!)^{-1} < (r!)^{-1}$. Then we combine the party knowledge XW ($X \in \mathbb{R}^{m \times n}$, and $W \in \mathbb{R}^{n \times k}$). From Theorem 1, XX^T is the only information required by SIM with respect to I_1 with the knowledge of XW . In other words, combining the XW gives no more information other than XX^T . Therefore, given an unknown n , and the probability of solving the $X < (r!)^{-1}$, we have the probability of solving W is also $< (r!)^{-1}$.

Then we continue to combine the party knowledge of $\{\sum X^l W^l |_{l \in S \setminus I}, \sum X^l (X^l)^T |_{l \in S \setminus I}\}$. They are the sum of real number matrices and the sum of non-negative real number matrices respectively. Given the adversary threshold $t < s - 1$, which means the number of honest local nodes (i.e. $|l|_{l \in S \setminus I}$) is ≥ 2 , we have a negligible probability to derive any $X^l W^l$ or $X^l (X^l)^T$ from their sum. In addition, d_l (the number of columns of X^l) is also unknown to I . Therefore, with the probability of solving the $X < (r!)^{-1}$, the probability of solving either X^l or W^l is also $< (r!)^{-1}$.

At last, we combine the party knowledge of $\{X^{l'} |_{l' \in I}, W^{l'} |_{l' \in I}\}$. First, they are the input of I , which are independent of $\{\sum X^l W^l |_{l \in S \setminus I}, \sum X^l (X^l)^T |_{l \in S \setminus I}\}$. Next, we combine them into $X(X^H)^T$, which will give the adversary the problem of solving $X^H (X^H)^T$. As each of $d_l |_{l \in S \setminus I}$ is unknown to I , we have the number of column of X^H is also unknown to I . Thus, with Lemma 1, we have the probability of solving the X^H of rank r is $\leq (r!)^{-1}$. Similarly, combining $\{X^{l'} |_{l' \in I}, W^{l'} |_{l' \in I}\}$ to XW will also give the probability of solving $W^H \leq (r!)^{-1}$.

Thus, PRIVCOLL preserves ε -privacy against adversary parties I on X , and $\varepsilon \leq (r!)^{-1}$.

With Theorem 2, we demonstrate that, with a sufficient rank of the training dataset of honest parties, e.g. ≥ 35 , which is common in real-world datasets, PRIVCOLL achieves 10^{-40} -privacy.

5 Correctness Analysis and Case Study

In this section, we first prove the correctness of PRIVCOLL when distributing learning algorithms that are based on gradient descent optimization. Then we use a recurrent neural network as a case study to illustrate the collaborative learning process in PRIVCOLL.

5.1 Correctness of PrivColl's Gradient Descent Optimization

The following theorem demonstrates that if a non-distributed gradient descent optimization algorithm taking X as input, denoted by $\mathcal{F}_{GD}(X)$, converges to a local/global minima η , then executing PRIVCOLL with the same hyper settings (such as cost function, step size, and model structure) on X^1, \dots, X^s , denoted by $\mathcal{F}'_{GD}(\{(S_l, X^l), \text{Agg}\}_{S_l \in S})$, also converges to η .

Theorem 3. *PRIVCOLL's distributed algorithm of solving gradient descent optimization $\mathcal{F}'_{GD}(\{(S_l, X^l), \text{Agg}\}_{S_l \in S})$ is correct.*

Proof. Let $\mathcal{F}_{GD}(X) = \eta$ denote the convergence of $\mathcal{F}_{GD}(X)$ to the local/global minima η . Let W_i denote the model parameters of \mathcal{F}_{GD} at i^{th} training iteration. Let $W'_i = |\{W_i^l\}|_{l \in \{1, \dots, s\}}$ denote the vertical concatenation on $\{W_i^l\}_{l \in \{1, \dots, s\}}$, i.e., the model parameters of \mathcal{F}'_{GD} at i^{th} training iteration.

In \mathcal{F}_{GD} , the i^{th} ($i \geq 1$) training iteration update W_i such that

$$\begin{aligned} W_i &= W_{i-1} - \alpha \frac{\partial J}{\partial W} = W_{i-1} - \alpha \frac{\partial J}{\partial (XW_{i-1})} \frac{\partial XW_{i-1}}{\partial W_{i-1}} \\ &= W_{i-1} - \alpha \frac{\partial J}{\partial (XW_{i-1})} X. \end{aligned} \quad (9)$$

In \mathcal{F}'_{GD} , each node S_l updates its local W_i^l in

$$\begin{aligned} W_i^l &= W_{i-1}^l - \alpha \Delta X^l = W_{i-1}^l - \alpha \frac{\partial J}{\partial (XW'_{i-1})} \frac{\partial X^l W'_{i-1}}{\partial W_{i-1}^l} \\ &= W_{i-1}^l - \alpha \frac{\partial J}{\partial (XW'_{i-1})} X^l. \end{aligned}$$

Since $W'_i = |\{W_i^l\}|_{l \in \{1, \dots, s\}}$, and $X = |\{X^l\}|_{l \in \{1, \dots, s\}}$, we have in \mathcal{F}'_{GD} that

$$W'_i = |\{(W_{i-1}^l - \alpha \frac{\partial J}{\partial (XW'_{i-1})} X^l)\}|_{l \in \{1, \dots, s\}} = W'_{i-1} - \alpha \frac{\partial J}{\partial (XW'_{i-1})} X. \quad (10)$$

Comparing Equation 9 and 10, we can find that W_i and W'_i are updated using the same equation. Therefore, with $\mathcal{F}_{GD}(X) = \eta$, the gradient descent guarantees $\mathcal{F}'_{GD}(\{(S_l, X^l), \text{Agg}\}_{S_l \in S})$ also converges to η .

Thus, $\mathcal{F}'_{GD}(\{(S_l, X^l), \text{Agg}\}_{S_l \in S}) = \mathcal{F}_{GD}(X)$.

5.2 Case Study

Figure 2 shows an example of a two-layer feed-forward recurrent neural network (RNN). Every neural layer is attached with a time subscript c . The weight matrix W maps the input vector $X^{(c)}$ to the hidden layer $h^{(c)}$. The weight matrix V propagates the hidden layer to the output layer $\hat{y}^{(c)}$. The weight matrix U maps the previous hidden layer to the current one.

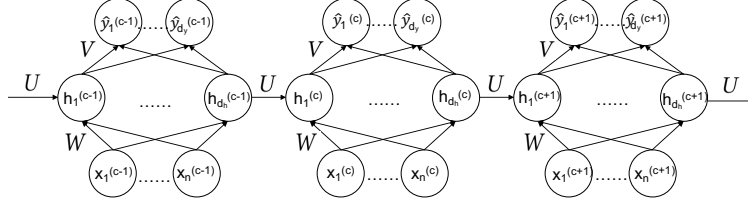


Fig. 2: An example of recurrent neural network

Original algorithm. Recall that the original non-distributed version of the RNN is divided into the forward propagation and backward propagation through time. First, in the forward propagation, the output of the hidden layer propagated from the input layer is calculated as

$$\begin{aligned} Z_h^{(c)} &= X^{(c)}W + Uh^{(c-1)} + b_h \\ h^{(c)} &= \sigma_1(Z_h^{(c)}) \end{aligned} \quad (11)$$

The output of the output layer propagated from hidden layer is calculated as

$$\begin{aligned} Z_y^{(c)} &= Vh^{(c)} + b_y \\ \hat{y}^{(c)} &= \sigma_2(Z_y^{(c)}) \end{aligned} \quad (12)$$

Then, the cost function $\sum_{c=0}^T J(\hat{y}^{(c)}, y^{(c)})$, and the coefficient matrices W, U, V are updated using the backward propagation through time.

The gradients of $J^{(c)}$ with respect to V is calculated as $\frac{\partial J^{(c)}}{\partial V} = \frac{\partial J^{(c)}}{\partial \hat{y}^{(c)}} \frac{\partial \hat{y}^{(c)}}{\partial Z_y^{(c)}} \frac{\partial Z_y^{(c)}}{\partial V}$. We let $\frac{\partial J^{(c)}}{\partial \hat{y}^{(c)}} = \delta_{loss}^{(c)}$, $\delta_y^{(c)} = \sigma_2'(Z_y^{(c)})$. The gradient of $J^{(c)}$ with respect to V can be written as:

$$\frac{\partial J^{(c)}}{\partial V} = [\delta_{loss}^{(c)} \circ \delta_y^{(c)}](h^{(c)})^T. \quad (13)$$

The gradients of $J^{(c)}$ with respect to U is calculated as $\frac{\partial J^{(c)}}{\partial U} = \frac{\partial J^{(c)}}{\partial \hat{y}^{(c)}} \frac{\partial \hat{y}^{(c)}}{\partial h^{(c)}} \frac{\partial h^{(c)}}{\partial U}$, where $\frac{\partial J^{(c)}}{\partial \hat{y}^{(c)}} \frac{\partial \hat{y}^{(c)}}{\partial h^{(c)}} = V^T[\delta_{loss}^{(c)} \circ \delta_y^{(c)}]$, and

$$\frac{\partial h^{(c)}}{\partial U} = \sum_{k=0}^c \frac{\partial h^{(c)}}{\partial h^{(k)}} \frac{\partial h^{(k)}}{\partial U} = \sum_{k=0}^c \left(\prod_{i=k+1}^c \sigma_1'(Z_h^{(i)})U \right) \sigma_1'(Z_h^{(k)})h^{(k-1)}.$$

Let $\delta_h^{(c)} = \sigma'_1(Z_h^{(c)})$, then the gradient of $J^{(c)}$ with respect to U is

$$\frac{\partial J^{(c)}}{\partial U} = V^T \delta_{loss}^{(c)} \circ \delta_{\hat{y}}^{(c)} \left(\sum_{k=0}^c \left(\prod_{i=k+1}^c \delta_h^{(i)} U \right) \delta_h^{(k)} h^{(k-1)} \right). \quad (14)$$

Similarly, the gradients of $J^{(c)}$ with respect to W is calculated as:

$$\frac{\partial J^{(c)}}{\partial W} = V^T \delta_{loss}^{(c)} \circ \delta_{\hat{y}}^{(c)} \left(\sum_{k=0}^c \left(\prod_{i=k+1}^c \delta_h^{(i)} U \right) \delta_h^{(k)} X^{(k)} \right). \quad (15)$$

Algorithm 1 Privacy Preserving Collaborative Recurrent Neural Network

```

1: Input: Local training data  $X^{l(c)}$  ( $l = [1, \dots, s], c = [0, \dots, T]$ ),
2:   learning rate  $\alpha$ 
3: Output: Model parameters  $W^l$  ( $l = [1, \dots, s]$ ),  $U, V$ 
4: Initialize: Randomize  $W^l$  ( $l = [1, \dots, s]$ ),  $U, V$ 
5: repeat
6:   for all  $S_l \in S$  do in parallel
7:      $S_l : X^{l(c)} W^l, c = [0, \dots, T]$ 
8:      $S_l : srt^{(c)} \leftarrow X^{l(c)} W^l, c = [0, \dots, T]$ 
9:      $\{(S_l, E_{srt}^{(c)})\}_{S_l \in S} \leftarrow Shr(srt^{(c)}, S)$ 
10:  end for
11:   $X^{(c)} W \leftarrow Rec(\{(S_l, E_{srt}^{(c)})\}_{S_l \in S})$ 
12:   $A : Z_h^{(c)} \leftarrow X^{(c)} W + U h^{(c-1)} + b_h, h^{(c)} \leftarrow \sigma_1(Z_h^{(c)})$ 
13:   $A : Z_y^{(c)} \leftarrow V h^{(c)} + b_y, \hat{y}^{(c)} = \sigma_2(Z_y^{(c)})$ 
14:   $A : \delta_{loss}^{(c)} \leftarrow \frac{\partial J^{(c)}}{\partial \hat{y}^{(c)}}, \delta_{\hat{y}}^{(c)} \leftarrow \sigma'_2(Z_y^{(c)}), \delta_h^{(c)} = \sigma'_1(Z_h^{(c)})$ 
15:   $A : \frac{\partial J^{(c)}}{\partial V} \leftarrow [\delta_{loss}^{(c)} \circ \delta_{\hat{y}}^{(c)}] (h^{(c)})^T$ 
16:   $A : \frac{\partial J^{(c)}}{\partial U} \leftarrow V^T \delta_{loss}^{(c)} \circ \delta_{\hat{y}}^{(c)} \left( \sum_{k=0}^c \left( \prod_{i=k+1}^c \delta_h^{(i)} U \right) \delta_h^{(k)} h^{(k-1)} \right)$ 
17:   $A : V \leftarrow V - \alpha \sum_{c=0}^T \frac{\partial J^{(c)}}{\partial V}$ 
18:   $A : U \leftarrow U - \alpha \sum_{c=0}^T \frac{\partial J^{(c)}}{\partial U}$ 
19:  for all  $S_l \in S$  do in parallel
20:     $\frac{\partial J^{(c)}}{\partial W^l} = V^T \delta_{loss}^{(c)} \circ \delta_{\hat{y}}^{(c)} \left( \sum_{k=0}^c \left( \prod_{i=k+1}^c \delta_h^{(i)} U \right) \delta_h^{(k)} X^{l(k)} \right)$ 
21:     $W^l \leftarrow W^l - \alpha \sum_{c=0}^T \frac{\partial J^{(c)}}{\partial W^l}$ 
22:  end for
23: until Convergence

```

PrivColl algorithm. In PRIVCOLL, each local node keeps $X^{l(c)}, c = [0, \dots, T]$, and maintains the coefficient matrix W^l locally. The aggregation node maintains coefficient matrices U, V . Similar to its original non-distributed counterpart, the

training process is divided into the forward propagation and backward propagation through time. Below we briefly outline these steps and the detailed algorithm is given by Algorithm 1.

In the forward propagation, local nodes compute $X^{l(c)}W^l, c = [0, \dots, T]$ respectively (line 7) (line number in Algorithm 1), and $X^{(c)}W = \sum_{l=1}^s X^{l(c)}W^l$ is calculated using the secret-sharing scheme (line 8 to line 11). The aggregation node then computes $Z_h^{(c)}, h^{(c)}, Z_y^{(c)}, \hat{y}^{(c)}$ using Equation 11 and 12 (line 12 and line 13).

In the backward propagation through time, for each $J^{(c)}$ at time t , the aggregation node computes $\delta_{loss}^{(c)}, \delta_{\hat{y}}^{(c)}, \delta_h^{(k)}, k = [0, \dots, t]$, and sends $\delta_{loss}^{(c)}, \delta_{\hat{y}}^{(c)}, \delta_h^{(k)}$ to local nodes (line 14). Then the aggregation node computes the gradients of $J^{(c)}$ with respect to V, U using Equation 13 and 14 (line 15 and line 16), and updates U, V (line 17 and line 18). The local nodes compute the gradients of $J^{(c)}$ with respect to W^l using Equation 16, and update W^l respectively (line 19 to line 22).

$$\frac{\partial J^{(c)}}{\partial W^l} = V^T \delta_{loss}^{(c)} \circ \delta_{\hat{y}}^{(c)} \left(\sum_{k=0}^c \left(\prod_{i=k+1}^c \delta_h^{(i)} U \right) \delta_h^{(k)} X^{l(k)} \right). \quad (16)$$

6 Performance Evaluation

We implement PRIVCOLL in C++. It uses the Eigen library [17] to handle matrix operations, and uses ZeroMQ library [21] to implement the distributed messaging. The experiments are executed on four Amazon EC2 c4.8xlarge machines with 60GB of RAM each, three of which act as local nodes and the other acts as the aggregation node. To simulate the real-world scenarios, we execute PRIVCOLL on both LAN and WAN network settings. In the LAN setting, machines are hosted in a same region, and the average network bandwidth is 1GB/s. In the WAN setting, we host these machines in different continents. The average network latency (one-way) is 137.7ms, and the average network throughput is 9.27MB/s. We collect 10 runs for each data point in the results and report the average. We use the MNIST dataset [27], and duplicate its samples when its size is less than the sample size m ($m \geq 60,000$).

We take non-private machine learning which trains on the concatenated dataset as the baseline, and compare with MZ17 [31], which is the state-of-the-art cryptographic solution for privacy preserving machine learning. It is based on oblivious transfer (MZ17-OT) and linearly homomorphic encryption (MZ17-LHE). As shown in Fig. 3, PRIVCOLL achieves significant efficiency improvement over MZ17, and due to parallelization in the computing of the local nodes, PRIVCOLL also outperforms the non-private baselines in the LAN network setting.

Linear regression and logistic regression. We use mini-batch stochastic gradient descent (SGD) for training the linear regression and logistic regression. We set the batch size $|B| = 40$ with 4 sample sizes (1,000-100,000) in the linear regression and logistic regression.

In the LAN setting, PRIVCOLL achieves around 45x faster than MZ17-OT. It takes 13.11s for linear regression (Figure 3a) and 12.73s for logistic regression (Figure 3b) with sample size $m = 100,000$, while in MZ17-OT, 594.95s and 605.95s are reported respectively. PRIVCOLL is also faster than the baseline, which takes 17.20s and 17.42s for linear/logistic regression respectively. In the WAN setting, PRIVCOLL is around 9x faster than MZ17-LHE. It takes 1408.75s for linear regression (Figure 3d) and 1424.94s for logistic regression (Figure 3e) with sample size $m = 100,000$, while in MZ17-LHE, it takes 12841.2s and 13441.2s respectively with the same sample size. It is worth mentioning that in MZ17, an MPC-friendly alternative function is specifically designed to replace non-linear sigmoid functions for training logistic regression, while in our framework, the non-linear function is used as usual. To further break down the overhead to computation and communication, we summarize the results of linear regression and logistic regression on other sample sizes in Table 1.

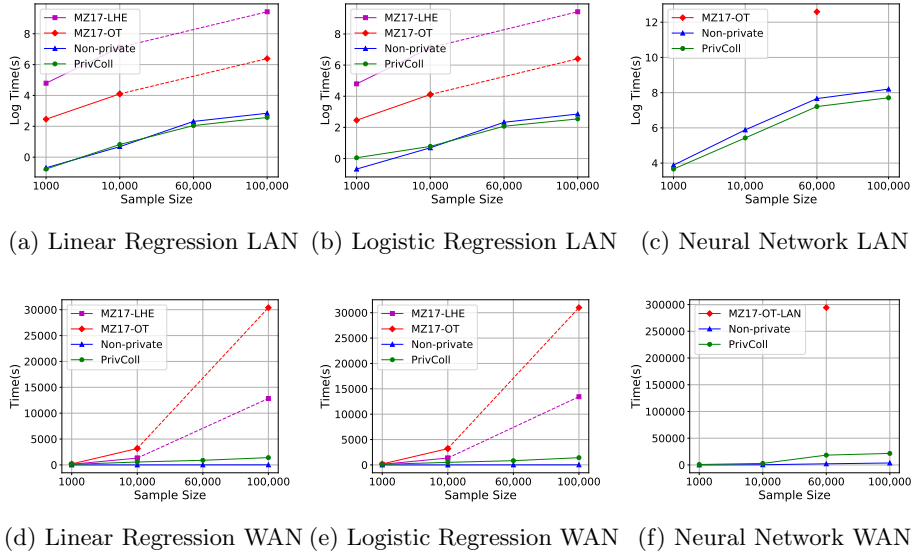


Fig. 3: Efficiency comparison. a-c) the natural logarithm of running time(s) as the sample size increases. d-f) running time(s) as the sample size increases.

Neural Network. We implement a fully connected neural network in PRIVCOLL. It has two hidden layers with 128 neurons in each layer (same as MZ17) and takes a sigmoid function as the activation function. For training the neural network, we set the batch size $|B| = 150$ with 4 sample sizes (1,000-60,000).

In the LAN network setting, PRIVCOLL achieves 1352.87s (around 22.5 minutes) (Figure 3c) with sample size $m = 60,000$, while in MZ17, it takes

Table 1: PRIVCOLL’s Overhead Breakdown for Linear/Logistic Regression

	Linear Regression						Logistic Regression					
	Computation	Communication		Total		Computation	Communication		Total			
		LAN	WAN	LAN	WAN		LAN	WAN	LAN	WAN		
m=1,000	0.445s	0.016s	103.28s	0.461s	103.73s	0.467s	0.583s	109.57s	1.050s	110.04s		
m=10,000	1.293s	0.978s	561.53s	2.271s	562.83s	1.368s	0.812s	506.64s	2.180s	508.01s		
m=60,000	6.155s	1.578s	887.63s	7.733s	893.79s	6.271s	1.683s	829.69s	7.954s	835.96s		
m=100,000	10.07s	3.037s	1398.67s	13.11s	1408.75s	10.29s	2.434s	1414.64s	12.73s	1424.94s		

294, 239.7s (more than 81 hours) with the same sample size. PRIVCOLL also outperforms the non-private baseline which takes 2, 127.87s. In the WAN setting, PRIVCOLL achieves 18, 367.88s (around 5.1 hours) with sample size $m = 60,000$ (Figure 3f), while in MZ17, it is not yet practical for training neural networks in WAN setting due to the high number of interactions and high communication. Note that, in Figure 3f, we still plot the MZ17-OT-LAN result (294, 239.7s), showing that even when running our framework in the WAN setting, it is still much more efficient compared to the MPC solutions in MZ17 run in the LAN setting. The overhead breakdown on computation and communication is summarized in Table 2.

Table 2: PRIVCOLL’s Overhead Breakdown for Neural Network

	Neural Network				
	Computation	Communication		Total	
		LAN	WAN	LAN	WAN
m=1,000	30.14s	8.729s	795.27s	38.87s	825.42s
m=10,000	223.08s	4.683s	2662.58s	227.76s	2885.66s
m=60,000	1320.77s	32.10s	17047.10s	1352.87s	18367.88s
m=100,000	2180.74s	47.99s	19364.73s	2228.73s	21545.47s

7 Related Work

The studies most related to PRIVCOLL are [23, 49]. Zheng et al [49] employ the lightweight additive secret sharing scheme for secure outsourcing of the decision tree algorithm for classification. Hu et al [23] propose FDML, which is a collaborative machine learning framework for distributed features, and the model parameters are protected by additive noise mechanism within the framework of differential privacy.

There also have been some previous research efforts which have explored collaborative learning without exposing their trained models [24, 33, 46, 47]. For example, Papernot et al [33] make use of transfer learning in combination with differential privacy to learn an ensemble of *teacher* models on data partitions, and then use these models to train a private *student* model.

In addition, there are more works on generic privacy-preserving machine learning frameworks via HE/MPC solutions [9, 14, 20, 25, 26, 34, 35, 41, 43, 45] or differential privacy mechanism [1, 2, 18, 39]. Recent studies [11, 12] propose a hybrid multi-party computation protocol for securely computing a linear regression model. In [28], an approach is proposed for transforming an existing neural network to an oblivious neural network supporting privacy-preserving predictions. In [43], a secure protocol is presented to calculate the delta function in the back-propagation training. In [31], a MPC-friendly alternative function is specifically designed to replace non-linear sigmoid and softmax functions, as the division and the exponentiation in these function are expensive to compute on shared values.

8 Conclusion

We have presented PRIVCOLL, a practical privacy-preserving collaborative machine learning framework. PRIVCOLL guarantees privacy preservation for both local training data and models trained on them, against an honest-but-curious adversary. It also ensures the correctness of a wide range of machine/deep learning algorithms, such as linear regression, logistic regression, and a variety of neural networks. Meanwhile, PRIVCOLL achieves a practical applicability. It is much more efficient compared to other state-of-art solutions.

References

1. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep learning with differential privacy. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. pp. 308–318 (2016)
2. Abuadbba, S., Kim, K., Kim, M., Thapa, C., Camtepe, S.A., Gao, Y., Kim, H., Nepal, S.: Can we use split learning on 1d cnn models for privacy preserving training? *arXiv preprint arXiv:2003.12365* (2020)
3. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic encryption security standard. Tech. rep., HomomorphicEncryption.org (2018)
4. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: *European Symposium on Research in Computer Security*. pp. 192–206. Springer (2008)
5. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. pp. 1175–1191. ACM (2017)
6. Canetti, R.: *Theory of cryptography*. Springer (2008)
7. Chen, Y.R., Rezapour, A., Tzeng, W.G.: Privacy-preserving ridge regression on distributed data. *Information Sciences* **451**, 34–49 (2018)
8. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* **9**(3–4), 211–407 (2014)

9. Esposito, C., Su, X., Aljawarneh, S.A., Choi, C.: Securing collaborative deep learning in industrial applications within adversarial scenarios. *IEEE Transactions on Industrial Informatics* **14**(11), 4972–4981 (2018)
10. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. pp. 1322–1333 (2015)
11. Gascón, A., Schoppmann, P., Balle, B., Raykova, M., Doerner, J., Zahur, S., Evans, D.: Secure linear regression on vertically partitioned datasets. *IACR Cryptology ePrint Archive* **2016**, 892 (2016)
12. Gascón, A., Schoppmann, P., Balle, B., Raykova, M., Doerner, J., Zahur, S., Evans, D.: Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies* **2017**(4), 345–364 (2017)
13. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. pp. 169–178 (2009)
14. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: *International Conference on Machine Learning*. pp. 201–210 (2016)
15. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pp. 307–328 (2019)
16. Golub, G., Van Loan, C.: *Matrix computations* 3rd edition the john hopkins university press. Baltimore, MD (1996)
17. Guennebaud, G., Jacob, B., et al.: *Eigen v3*. <http://eigen.tuxfamily.org> (2010)
18. Gupta, O., Raskar, R.: Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* **116**, 1–8 (2018)
19. Hagestedt, I., Zhang, Y., Humbert, M., Berrang, P., Tang, H., Wang, X., Backes, M.: Mbeacon: Privacy-preserving beacons for dna methylation data. In: *NDSS* (2019)
20. Hardy, S., Henecka, W., Ivey-Law, H., Nock, R., Patrini, G., Smith, G., Thorne, B.: Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677* (2017)
21. Hintjens, P.: *ZeroMQ: messaging for many applications*. ” O’Reilly Media, Inc.” (2013)
22. Horn, R.A., Johnson, C.R.: *Matrix analysis*. Cambridge university press (2012)
23. Hu, Y., Niu, D., Yang, J., Zhou, S.: Fdml: A collaborative machine learning framework for distributed features. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. pp. 2232–2240 (2019)
24. Jia, Q., Guo, L., Jin, Z., Fang, Y.: Privacy-preserving data classification and similarity evaluation for distributed systems. In: *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. pp. 690–699. IEEE (2016)
25. Ko, R.K., Russello, G., Nelson, R., Pang, S., Cheang, A., Dobbie, G., Sarrafzadeh, A., Chaisiri, S., Asghar, M.R., Holmes, G.: Stratus: Towards returning data control to cloud users. In: *International Conference on Algorithms and Architectures for Parallel Processing*. pp. 57–70. Springer (2015)
26. Kwabena, O.A., Qin, Z., Zhuang, T., Qin, Z.: Mscryptonet: Multi-scheme privacy-preserving deep learning in cloud computing. *IEEE Access* **7**, 29344–29354 (2019)
27. LeCun, Y., Cortes, C.: *MNIST handwritten digit database* (2010), <http://yann.lecun.com/exdb/mnist/>

28. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. pp. 619–631 (2017)
29. Marc, T., Stopar, M., Hartman, J., Bizjak, M., Modic, J.: Privacy-enhanced machine learning with functional encryption. In: *European Symposium on Research in Computer Security*. pp. 3–21. Springer (2019)
30. Melis, L., Song, C., De Cristofaro, E., Shmatikov, V.: Exploiting unintended feature leakage in collaborative learning. In: *2019 IEEE Symposium on Security and Privacy (SP)*. pp. 691–706. IEEE (2019)
31. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: *2017 IEEE Symposium on Security and Privacy (SP)*. pp. 19–38. IEEE (2017)
32. Papernot, N., McDaniel, P., Sinha, A., Wellman, M.: Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814* (2016)
33. Papernot, N., Song, S., Mironov, I., Raghunathan, A., Talwar, K., Erlingsson, Ú.: Scalable private learning with pate. *arXiv preprint arXiv:1802.08908* (2018)
34. Ryffel, T., Trask, A., Dahl, M., Wagner, B., Mancuso, J., Rueckert, D., Passerat-Palmbach, J.: A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017* (2018)
35. Sadat, M.N., Aziz, M.M.A., Mohammed, N., Chen, F., Wang, S., Jiang, X.: Safety: Secure gwas in federated environment through a hybrid solution with intel sgx and homomorphic encryption. *arXiv preprint arXiv:1703.02577* (2017)
36. Sharma, S., Chen, K.: Confidential boosting with random linear classifiers for out-sourced user-generated data. In: *European Symposium on Research in Computer Security*. pp. 41–65. Springer (2019)
37. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. pp. 1310–1321 (2015)
38. Song, S., Chaudhuri, K., Sarwate, A.D.: Stochastic gradient descent with differentially private updates. In: *2013 IEEE Global Conference on Signal and Information Processing*. pp. 245–248. IEEE (2013)
39. Vepakomma, P., Gupta, O., Swedish, T., Raskar, R.: Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564* (2018)
40. Wang, S., Pi, A., Zhou, X.: Scalable distributed dl training: Batching communication and computation. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. vol. 33, pp. 5289–5296 (2019)
41. Will, M.A., Nicholson, B., Tiehuis, M., Ko, R.K.: Secure voting in the cloud using homomorphic encryption and mobile agents. In: *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*. pp. 173–184. IEEE (2015)
42. Yao, A.C.C.: How to generate and exchange secrets. In: *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. pp. 162–167. IEEE (1986)
43. Yuan, J., Yu, S.: Privacy preserving back-propagation neural network learning made practical with cloud computing. *IEEE Transactions on Parallel and Distributed Systems* **25**(1), 212–221 (2014)
44. Zhang, J., Chen, B., Yu, S., Deng, H.: Pefl: A privacy-enhanced federated learning scheme for big data analytics. In: *2019 IEEE Global Communications Conference (GLOBECOM)*. pp. 1–6. IEEE (2019)
45. Zhang, X., Ji, S., Wang, H., Wang, T.: Private, yet practical, multiparty deep learning. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. pp. 1442–1452. IEEE (2017)

46. Zhang, Y., Bai, G., Zhong, M., Li, X., Ko, R.: Differentially private collaborative coupling learning for recommender systems. *IEEE Intelligent Systems* (2020)
47. Zhang, Y., Zhao, X., Li, X., Zhong, M., Curtis, C., Chen, C.: Enabling privacy-preserving sharing of genomic data for gwass in decentralized networks. In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. pp. 204–212. ACM (2019)
48. Zheng, H., Ye, Q., Hu, H., Fang, C., Shi, J.: Bdpl: A boundary differentially private layer against machine learning model extraction attacks. In: *European Symposium on Research in Computer Security*. pp. 66–83. Springer (2019)
49. Zheng, Y., Duan, H., Wang, C.: Towards secure and efficient outsourcing of machine learning classification. In: *European Symposium on Research in Computer Security*. pp. 22–40. Springer (2019)