A Scalable Approach for Privacy-Preserving Collaborative Machine Learning

J. So
ECE Department
University of Southern California (USC)
jinhyuns@usc.edu

B. Guler
ECE Department
University of California, Riverside
bguler@ece.ucr.edu

A.S. Avestimehr
ECE Department
University of Southern California (USC)
avestimehr@ee.usc.edu

Abstract

We consider a collaborative learning scenario in which multiple data-owners wish to jointly train a logistic regression model, while keeping their individual datasets private from the other parties. We propose COPML, a fully-decentralized training framework that achieves scalability and privacy-protection simultaneously. The key idea of COPML is to securely encode the individual datasets to distribute the computation load effectively across many parties and to perform the training computations as well as the model updates in a distributed manner on the securely encoded data. We provide the privacy analysis of COPML and prove its convergence. Furthermore, we experimentally demonstrate that COPML can achieve significant speedup in training over the benchmark protocols. Our protocol provides strong statistical privacy guarantees against colluding parties (adversaries) with unbounded computational power, while achieving up to $16\times$ speedup in the training time against the benchmark protocols.

1 Introduction

Machine learning applications can achieve significant performance gains by training on large volumes of data. In many applications, the training data is distributed across multiple data-owners, such as patient records at multiple medical institutions, and furthermore contains sensitive information, e.g., genetic information, financial transactions, and geolocation information. Such settings give rise to the following key problem that is the focus of this paper: *How can multiple data-owners jointly train a machine learning model while keeping their individual datasets private from the other parties?*

More specifically, we consider a distributed learning scenario in which N data-owners (clients) wish to train a logistic regression model jointly without revealing information about their individual datasets to the other parties, even if up to T out of N clients collude. Our focus is on the semi-honest adversary setup, where the corrupted parties follow the protocol but may leak information in an attempt to learn the training dataset. To address this challenge, we propose a novel framework, COPML[1], that enables fast and privacy-preserving training by leveraging information and coding theory principles. COPML has three salient features:

- speeds up the training time significantly, by distributing the computation load effectively across a large number of parties,

[1] COPML stands for collaborative privacy-preserving machine learning.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

- advances the state-of-the-art privacy-preserving training setups by scaling to a large number of parties, as it can distribute the computation load effectively as more parties are added in the system,
- utilizes coding theory principles to secret share the dataset and model parameters which can significantly reduce the communication overhead and the complexity of distributed training.

At a high level, COPML can be described as follows. Initially, the clients secret share their individual datasets with the other parties, after which they carry out a secure multi-party computing (MPC) protocol to *encode* the dataset. This encoding operation transforms the dataset into a *coded* form that enables faster training and simultaneously guarantees privacy (in an information-theoretic sense). Training is performed over the encoded data via gradient descent. The parties perform the computations over the encoded data *as if they were computing over the uncoded dataset*. That is, the structure of the computations are the same for computing over the uncoded dataset versus computing over the encoded dataset. At the end of training, each client should only learn the final model, and no information should be leaked (in an information-theoretic sense) about the individual datasets or the intermediate model parameters, beyond the final model.

We characterize the theoretical performance guarantees of COPML, in terms of convergence, scalability, and privacy protection. Our analysis identifies a trade-off between privacy and parallelization, such that, each additional client can be utilized either for more privacy, by protecting against a larger number of collusions T, or more parallelization, by reducing the computation load at each client. Furthermore, we empirically demonstrate the performance of COPML by comparing it with cryptographic benchmarks based on secure multi-party computing (MPC) [39, 4, 3, 12], that can also be applied to enable privacy-preserving machine learning tasks (e.g. see [30, 14, 28, 25, 10, 8, 37, 27]). Given our focus on information-theoretic privacy, the most relevant MPC-based schemes for empirical comparison are the protocols from [4] and [3, 12] based on Shamir's secret sharing [33]. While several more recent works have considered MPC-based learning setups with information-theoretic privacy [37, 27], their constructions are limited to three or four parties.

We run extensive experiments over the Amazon EC2 cloud platform to empirically demonstrate the performance of COPML. We train a logistic regression model for image classification over the CIFAR-10 [23] and GISETTE [18] datasets. The training computations are distributed to up to $N = 50$ parties. We demonstrate that COPML can provide significant speedup in the training time against the state-of-the-art MPC baseline (up to $16.4\times$), while providing comparable accuracy to conventional logistic regression. This is primarily due to the parallelization gain provided by our system, which can distribute the workload effectively across many parties.

Other related works. Other than MPC-based setups, one can consider two notable approaches. The first one is Homomorphic Encryption (HE) [15], which enables computations on encrypted data, and has been applied to privacy-preserving machine learning [16, 20, 17, 41, 24, 22, 38, 19]. The privacy protection of HE depends on the size of the encrypted data, and computing in the encrypted domain is computationally intensive. The second approach is differential privacy (DP), which is a noisy release mechanism to protect the privacy of personally identifiable information. The main application of DP in machine learning is when the model is to be released publicly after training, so that individual data points cannot be backtracked from the released model [7, 34, 1, 31, 26, 32, 21]. On the other hand, our focus is on ensuring privacy during training, while preserving the accuracy of the model.

2 Problem Setting

We consider a collaborative learning scenario in which the training dataset is distributed across N clients. Client $j \in [N]$ holds an individual dataset denoted by a matrix $\mathbf{X}_j \in \mathbb{R}^{m_j \times d}$ consisting of m_j data points with d features, and the corresponding labels are given by a vector $\mathbf{y}_j \in \{0,1\}^{m_j}$. The overall dataset is denoted by $\mathbf{X} = [\mathbf{X}_1^\top, \ldots, \mathbf{X}_N^\top]^\top$ consisting of $m \triangleq \sum_{j\in[N]} m_j$ data points with d features, and corresponding labels $\mathbf{y} = [\mathbf{y}_1^\top, \ldots, \mathbf{y}_N^\top]^\top$, which consists of N individual datasets each one belonging to a different client. The clients wish to jointly train a logistic regression model $\mathbf{w}$ over the training set $\mathbf{X}$ with labels $\mathbf{y}$, by minimizing a cross entropy loss function,

$$C(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} \left(-y_i \log \hat{y}_i - (1-y_i) \log(1-\hat{y}_i)\right) \tag{1}$$

where $\hat{y}_i = g(\mathbf{x}_i \cdot \mathbf{w}) \in (0,1)$ is the probability of label i being equal to 1, $\mathbf{x}_i$ is the i^{th} row of matrix $\mathbf{X}$, and $g(\cdot)$ denotes the sigmoid function $g(z) = 1/(1+e^{-z})$. The training is performed through gradient descent, by updating the model parameters in the opposite direction of the gradient,

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta}{m}\mathbf{X}^\top(g(\mathbf{X} \times \mathbf{w}^{(t)}) - \mathbf{y}) \tag{2}$$

where $\nabla C(\mathbf{w}) = \frac{1}{m}\mathbf{X}^\top(g(\mathbf{X} \times \mathbf{w}) - \mathbf{y})$ is the gradient for (1), $\mathbf{w}^{(t)}$ holds the estimated parameters from iteration t, η is the learning rate, and function $g(\cdot)$ acts element-wise over the vector $\mathbf{X} \times \mathbf{w}^{(t)}$.

During training, the clients wish to protect the privacy of their individual datasets from other clients, even if up to T of them collude, where T is the *privacy parameter* of the system. There is no trusted party who can collect the datasets in the clear and perform the training. Hence, the training protocol should preserve the privacy of the individual datasets against any collusions between up to T adversarial clients. More specifically, this condition states that the adversarial clients should not learn any information about the datasets of the benign clients beyond what can already be inferred from the adversaries' own datasets.

To do so, client $j \in [N]$ initially secret shares its individual dataset $\mathbf{X}_j$ and $\mathbf{y}_j$ with the other parties. Next, clients carry out a secure MPC protocol to encode the dataset by using the received secret shares. In this phase, the dataset $\mathbf{X}$ is first partitioned into K submatrices $\mathbf{X} = [\mathbf{X}_1^\top, \cdots, \mathbf{X}_K^\top]^\top$ for some $K \in \mathbb{N}$. Parameter K characterizes the computation load at each client. Specifically, our system ensures that the computation load (in terms of gradient computations) at each client is equal to processing only $(1/K)^{th}$ of the entire dataset $\mathbf{X}$. The clients then encode the dataset by combining the K submatrices together with some randomness to preserve privacy. At the end of this phase, client $i \in [N]$ learns an encoded dataset $\widetilde{\mathbf{X}}_i$, whose size is equal to $(1/K)^{th}$ of the dataset $\mathbf{X}$. This process is only performed once for the dataset $\mathbf{X}$.

At each iteration of training, clients also encode the current estimation of the model parameters $\mathbf{w}^{(t)}$ using a secure MPC protocol, after which client $i \in [N]$ obtains the encoded model $\widetilde{\mathbf{w}}_i^{(t)}$. Client $i \in [N]$ then computes a local gradient $\widetilde{\mathbf{X}}_i^\top g(\widetilde{\mathbf{X}}_i \times \widetilde{\mathbf{w}}_i^{(t)})$ over the encoded dataset $\widetilde{\mathbf{X}}_i$ and encoded model $\widetilde{\mathbf{w}}_i^{(t)}$. After this step, clients carry out another secure MPC protocol to decode the gradient $\mathbf{X}^\top g(\mathbf{X} \times \mathbf{w}^{(t)})$ and update the model according to (2). As the decoding and model updates are performed using a secure MPC protocol, clients do not learn any information about the actual gradients or the updated model. In particular, client $i \in [N]$ only learns a secret share of the updated model, denoted by $[\mathbf{w}^{(t+1)}]_i$. Using the secret shares $[\mathbf{w}^{(t+1)}]_i$, clients $i \in [N]$ encode the model $\mathbf{w}^{(t+1)}$ for the next iteration, after which client i learns an encoded model $\widetilde{\mathbf{w}}_i^{(t+1)}$. Figure 1 demonstrates our system architecture.

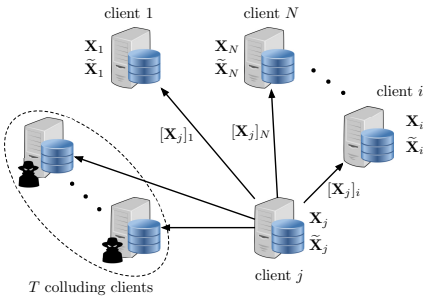


Figure 1: The multi-client distributed training setup with N clients. Client $j \in [N]$ holds a dataset $\mathbf{X}_j$ with labels $\mathbf{y}_j$. At the beginning of training, client j secret shares $\mathbf{X}_j$ and $\mathbf{y}_j$ to guarantee their information-theoretic privacy against any collusions between up to T clients. The secret share of $\mathbf{X}_j$ and $\mathbf{y}_j$ assigned from client j to client i is represented by $[\mathbf{X}_j]_i$ and $[\mathbf{y}_j]_i$, respectively.

3 The COPML Framework

COPML consists of four main phases: quantization; encoding and secret sharing; polynomial approximation; decoding and model update, as demonstrated in Figure 2. In the first phase, quantization, each client converts its own dataset from the real domain to finite field. In the second phase, clients create a secret share of their quantized datasets and carry out a secure MPC protocol to encode the datasets. At each iteration, clients also encode and create a secret share of the model parameters. In the third phase, clients perform local gradient computations over the encoded datasets and encoded model parameters by approximating the sigmoid function with a polynomial. Then, in the last phase, clients decode the local computations and update the model parameters using a secure MPC protocol. This process is repeated until the convergence of the model parameters.

Phase 1: Quantization. Computations involving secure MPC protocols are bound to finite field operations, which requires the representation of real-valued data points in a finite field $\mathbb{F}$. To do so, each client initially quantizes its dataset from the real domain to the domain of integers, and then

embeds it in a field $\mathbb{F}_p$ of integers modulo a prime p. Parameter p is selected to be sufficiently large to avoid wrap-around in computations. For example, in a 64-bit implementation with the CIFAR-10 dataset, we select $p = 2^{26} - 5$. The details of the quantization phase are provided in Appendix A.1.

Phase 2: Encoding and secret sharing. In this phase, client $j \in [N]$ creates a secret share of its quantized dataset $\mathbf{X}_j$ designated for each client $i \in [N]$ (including client j itself). The secret shares are constructed via Shamir's secret sharing with threshold T [33], to protect the privacy of the individual datasets against any collusions between up to T clients. To do so, client j creates a random polynomial, $h_j(z) = \mathbf{X}_j + z\mathbf{R}_{j1} + \ldots + z^T\mathbf{R}_{jT}$ where $\mathbf{R}_{ji}$ for $i \in [T]$ are i.i.d. uniformly distributed random matrices, and selects N distinct evaluation points $\lambda_1, \ldots, \lambda_N$ from $\mathbb{F}_p$. Then, client j sends client $i \in [N]$ a secret share $[\mathbf{X}_j]_i \triangleq h_j(\lambda_i)$ of its dataset $\mathbf{X}_j$. Client j also sends a secret share of its labels $\mathbf{y}_j$ to client $i \in [N]$, denoted by $[\mathbf{y}_j]_i$. Finally, the model is initialized randomly within a secure MPC protocol between the clients, and at the end client $i \in [N]$ obtains a secret share $[\mathbf{w}^{(0)}]_i$ of the initial model $\mathbf{w}^{(0)}$.

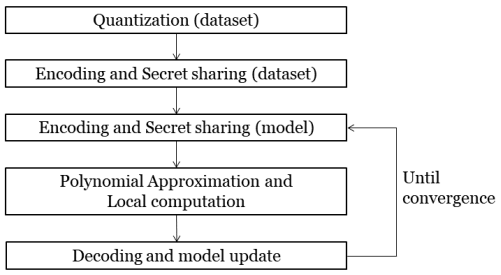


Figure 2: Flowchart of COPML.

After obtaining the secret shares $[\mathbf{X}_j]_i$ for $j \in [N]$, clients $i \in [N]$ encode the dataset using a secure MPC protocol and transform it into a *coded* form, which speeds up the training by distributing the computation load of gradient evaluations across the clients. Our encoding strategy utilizes Lagrange coding from [40][2], which has been applied to other problems such as privacy-preserving offloading of a training task [36] and secure federated learning [35]. However, we encode (and later decode) the secret shares of the datasets and not their true values. Therefore, clients do not learn any information about the true value of the dataset $\mathbf{X}$ during the encoding-decoding process.

The individual steps of the encoding process are as follows. Initially, the dataset $\mathbf{X}$ is partitioned into K submatrices $\mathbf{X} = [\mathbf{X}_1^\top, \ldots, \mathbf{X}_K^\top]^\top$ where $\mathbf{X}_k \in \mathbb{F}_p^{\frac{m}{K} \times d}$ for $k \in [K]$. To do so, client $i \in [N]$ locally concatenates $[\mathbf{X}_j]_i$ for $j \in [N]$ and partitions it into K parts, $[\mathbf{X}_k]_i$ for $k \in [K]$. Since this operation is done over the secret shares, clients do not learn any information about the original dataset $\mathbf{X}$. Parameter K quantifies the computation load at each client, as will be discussed in Section 4.

The clients agree on $K + T$ distinct elements $\{\beta_k\}_{k \in [K+T]}$ and N distinct elements $\{\alpha_i\}_{i \in [N]}$ from $\mathbb{F}_p$ such that $\{\alpha_i\}_{i \in [N]} \cap \{\beta_k\}_{k \in [K+T]} = \varnothing$. Client $i \in [N]$ then encodes the dataset using a Lagrange interpolation polynomial $u : \mathbb{F}_p \to \mathbb{F}_p^{\frac{m}{K} \times d}$ with degree at most $K + T - 1$,

$$[u(z)]_i \triangleq \sum_{k \in [K]} [\mathbf{X}_k]_i \cdot \prod_{l \in [K+T] \setminus \{k\}} \frac{z - \beta_l}{\beta_k - \beta_l} + \sum_{k=K+1}^{K+T} [\mathbf{Z}_k]_i \cdot \prod_{l \in [K+T] \setminus \{k\}} \frac{z - \beta_l}{\beta_k - \beta_l}, \tag{3}$$

where $[u(\beta_k)]_i = [\mathbf{X}_k]_i$ for $k \in [K]$ and $i \in [N]$. The matrices $\mathbf{Z}_k$ are generated uniformly at random[3] from $\mathbb{F}_p^{\frac{m}{K} \times d}$ and $[\mathbf{Z}_k]_i$ is the secret share of $\mathbf{Z}_k$ at client i. $[\mathbf{Z}_k]_i$ is the secret share of $\mathbf{Z}_k$ at client i. Client $i \in [N]$ then computes and sends $[\widetilde{\mathbf{X}}_j]_i \triangleq [u(\alpha_j)]_i$ to client $j \in [N]$. Upon receiving $\{[\widetilde{\mathbf{X}}_j]_i\}_{i \in [N]}$, client $j \in [N]$ can recover the encoded matrix $\widetilde{\mathbf{X}}_j$.[4] The role of $\mathbf{Z}_k$'s are to mask the dataset so that the encoded matrices $\widetilde{\mathbf{X}}_j$ reveal no information about the dataset $\mathbf{X}$, even if up to T clients collude, as detailed in Section 4.

Using the secret shares $[\mathbf{X}_j]_i$ and $[\mathbf{y}_j]_i$, clients $i \in [N]$ also compute $\mathbf{X}^T\mathbf{y} = \sum_{j \in [N]} \mathbf{X}_j^T \mathbf{y}_j$ using a secure multiplication protocol (see Appendix A.3 for details). At the end of this step, clients learn a secret share of $\mathbf{X}^T\mathbf{y}$, which we denote by $[\mathbf{X}^T\mathbf{y}]_i$ for client $i \in N$.

[2]Encoding of Lagrange coded computing is the same as a packed secret sharing [13].

[3]The random parameters can be generated by a crypto-service provider in an offline manner, or by using pseudo-random secret sharing [9].

[4]In fact, gathering only $T + 1$ secret shares is sufficient to recover $\widetilde{\mathbf{X}}_i$, due to the construction of Shamir's secret sharing [33]. Using this fact, one can speed up the execution by dividing the N clients into subgroups of $T + 1$ and performing the encoding locally within each subgroup. We utilize this property in our experiments.

At iteration t, client i initially holds a secret share of the current model, $[\mathbf{w}^{(t)}]_i$, and then encodes the model via a Lagrange interpolation polynomial $v : \mathbb{F}_p \to \mathbb{F}_p^d$ with degree at most $K+T-1$,

$$[v(z)]_i \triangleq \sum_{k\in[K]} [\mathbf{w}^{(t)}]_i \cdot \prod_{l\in[K+T]\setminus\{k\}} \frac{z-\beta_l}{\beta_k-\beta_l} + \sum_{k=K+1}^{K+T} [\mathbf{v}_k^{(t)}]_i \cdot \prod_{l\in[K+T]\setminus\{k\}} \frac{z-\beta_l}{\beta_k-\beta_l}, \tag{4}$$

where $[v(\beta_k)]_i = [\mathbf{w}^{(t)}]_i$ for $k \in [K]$ and $i \in [N]$. The vectors $\mathbf{v}_k^{(t)}$ are generated uniformly at random from $\mathbb{F}_p^d$. Client $i \in [N]$ then sends $[\widetilde{\mathbf{w}}_j^{(t)}]_i \triangleq [v(\alpha_j)]_i$ to client $j \in [N]$. Upon receiving $\{[\widetilde{\mathbf{w}}_j^{(t)}]_i\}_{i\in[N]}$, client $j \in [N]$ recovers the encoded model $\widetilde{\mathbf{w}}_j^{(t)}$.

Phase 3: Polynomial Approximation and Local Computations. Lagrange encoding can be used to compute polynomial functions only, whereas the gradient computations in (2) are not polynomial operations due to the sigmoid function. To this end, we approximate the sigmoid with a polynomial,

$$\hat{g}(z) = \sum_{i=0}^{r} c_i z^i, \tag{5}$$

where r and c_i represent the degree and coefficients of the polynomial, respectively. The coefficients are evaluated by fitting the sigmoid to the polynomial function via least squares estimation. Using this polynomial approximation, we rewrite the model update from (2) as,

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta}{m}\mathbf{X}^\top(\hat{g}(\mathbf{X}\times\mathbf{w}^{(t)}) - \mathbf{y}). \tag{6}$$

Client $i \in [N]$ then locally computes the gradient over the encoded dataset, by evaluating a function,

$$f(\widetilde{\mathbf{X}}_i, \widetilde{\mathbf{w}}_i^{(t)}) = \widetilde{\mathbf{X}}_i^\top \hat{g}(\widetilde{\mathbf{X}}_i \times \widetilde{\mathbf{w}}_i^{(t)}) \tag{7}$$

and secret shares the result with the other clients, by sending a secret share of (7), $[f(\widetilde{\mathbf{X}}_i, \widetilde{\mathbf{w}}_i^{(t)})]_j$, to client $j \in [N]$. At the end of this step, client j holds the secret shares $[f(\widetilde{\mathbf{X}}_i, \widetilde{\mathbf{w}}_i^{(t)})]_j$ corresponding to the local computations from clients $i \in [N]$. Note that (7) is a polynomial function evaluation in the finite field arithmetic and the degree of function f is $\deg(f) = 2r+1$.

Phase 4: Decoding and Model Update. In this phase, clients perform the decoding of the gradient using a secure MPC protocol, through polynomial interpolation over the secret shares $[f(\widetilde{\mathbf{X}}_i, \widetilde{\mathbf{w}}_i^{(t)})]_j$. The minimum number of clients needed for the decoding operation to be successful, which we call the *recovery threshold* of the protocol, is equal to $(2r+1)(K+T-1)+1$. In order to show this, we first note that, from the definition of Lagrange polynomials in (3) and (4), one can define a univariate polynomial $h(z) = f\big(u(z), v(z)\big)$ such that

$$h(\beta_i) = f\big(u(\beta_i), v(\beta_i)\big) = f\big(\mathbf{X}_i, \mathbf{w}^{(t)}\big) = \mathbf{X}_i^\top \hat{g}(\mathbf{X}_i \times \mathbf{w}^{(t)}) \tag{8}$$

for $i \in [K]$. Moreover, from (7), we know that client i performs the following computation,

$$h(\alpha_i) = f\big(u(\alpha_i), v(\alpha_i)\big) = f\big(\widetilde{\mathbf{X}}_i, \widetilde{\mathbf{w}}_i^{(t)}\big) = \widetilde{\mathbf{X}}_i^\top \hat{g}(\widetilde{\mathbf{X}}_i \times \widetilde{\mathbf{w}}_i^{(t)}). \tag{9}$$

The decoding process is based on the intuition that, the computations from (9) can be used as evaluation points $h(\alpha_i)$ to interpolate the polynomial $h(z)$. Since the degree of the polynomial $h(z)$ is $\deg\big(h(z)\big) \leq (2r+1)(K+T-1)$, all of its coefficients can be determined as long as there are at least $(2r+1)(K+T-1)+1$ evaluation points available. After $h(z)$ is recovered, the computation results in (8) correspond to $h(\beta_i)$ for $i \in [K]$.

Our decoding operation corresponds to a finite-field polynomial interpolation problem. More specifically, upon receiving the secret shares of the local computations $[f(\widetilde{\mathbf{X}}_j, \widetilde{\mathbf{w}}_j^{(t)})]_i$ from at least $(2r+1)(K+T-1)+1$ clients, client i locally computes

$$[f(\mathbf{X}_k, \mathbf{w}^{(t)})]_i \triangleq \sum_{j\in\mathcal{I}_i} [f(\widetilde{\mathbf{X}}_j, \widetilde{\mathbf{w}}_j^{(t)})]_i \cdot \prod_{l\in\mathcal{I}_i\setminus\{j\}} \frac{\beta_k-\alpha_l}{\alpha_j-\alpha_l} \tag{10}$$

for $k \in [K]$, where $\mathcal{I}_i \subseteq [N]$ denotes the set of the $(2r+1)(K+T-1)+1$ fastest clients who send their secret share $[f(\widetilde{\mathbf{X}}_j, \widetilde{\mathbf{w}}_j^{(t)})]_i$ to client i.

After this step, client i locally aggregates its secret shares $[f(\mathbf{X}_k, \mathbf{w}^{(t)})]_i$ to compute $\sum_{k=1}^{K}[f(\mathbf{X}_k, \mathbf{w}^{(t)})]_i$, which in turn is a secret share of $\mathbf{X}^T\,\hat{g}(\mathbf{X}\times\mathbf{w}^{(t)})$ since,

$$\sum_{k=1}^{K} f(\mathbf{X}_k, \mathbf{w}^{(t)}) = \sum_{k=1}^{K} \mathbf{X}_k^\top\,\hat{g}(\mathbf{X}_k\times\mathbf{w}^{(t)}) = \mathbf{X}^\top\,\hat{g}(\mathbf{X}\times\mathbf{w}^{(t)}). \tag{11}$$

Let $[\mathbf{X}^\top\,\hat{g}(\mathbf{X}\times\mathbf{w}^{(t)})]_i \triangleq \sum_{k=1}^{K}[f(\mathbf{X}_k, \mathbf{w}^{(t)})]_i$ denote the secret share of (11) at client i. Client i then computes $[\mathbf{X}^\top\,\hat{g}(\mathbf{X}\times\mathbf{w}^{(t)})]_i - [\mathbf{X}^\top\,\mathbf{y}]_i$, which in turn is a secret share of the gradient $\mathbf{X}^\top(\hat{g}(\mathbf{X}\times\mathbf{w}^{(t)}) - \mathbf{y})$. Since the decoding operations are carried out using the secret shares, at the end of the decoding process, the clients only learn a secret share of the gradient and not its true value.

Next, clients update the model according to (6) using a secure MPC protocol, using the secret shared model $[\mathbf{w}^{(t)}]_i$ and the secret share of the gradient $[\mathbf{X}^\top\,\hat{g}(\mathbf{X}\times\mathbf{w}^{(t)})]_i - [\mathbf{X}^\top\,\mathbf{y}]_i$. A major challenge in performing the model update in (6) in the finite field is the multiplication with parameter $\frac{\eta}{m}$, where $\frac{\eta}{m} < 1$. In order to perform this operation in the finite field, one potential approach is to treat it as a computation on integer numbers and preserve full accuracy of the results. This in turn requires a very large field size as the range of results grows exponentially with the number of multiplications, which becomes quickly impractical as the number of iterations increase [28]. Instead, we address this problem by leveraging the secure truncation technique from [6]. This protocol takes secret shares $[a]_i$ of a variable a as input as well as two public integer parameters k_1 and k_2 such that $a \in \mathbb{F}_{2^{k_2}}$ and $0 < k_1 < k_2$. The protocol then returns the secret shares $[z]_i$ for $i \in [N]$ such that $z = \lfloor \frac{a}{2^{k_1}} \rfloor + s$ where s is a random bit with probability $P(s = 1) = (a \mod 2^{k_1})/(2^{k_1})$. Accordingly, the protocol rounds $a/(2^{k_1})$ to the closest integer with probability $1 - \tau$, with τ being the distance between $a/(2^{k_1})$ and that integer. The truncation operation ensures that the range of the updated model always stays within the range of the finite field.

Since the model update is carried out using a secure MPC protocol, at the end of this step, client $i \in [N]$ learns only a secret share $[\mathbf{w}^{(t+1)}]_i$ of the updated model $\mathbf{w}^{(t+1)}$, and not its actual value. In the next iteration, using $[\mathbf{w}^{(t+1)}]_i$, client $i \in [N]$ locally computes $[\widetilde{\mathbf{w}}_j^{(t+1)}]_i$ from (4) and sends it to client $j \in [N]$. Client j then recovers the encoded model $\widetilde{\mathbf{w}}_j^{(t+1)}$, which is used to compute (7).

The implementation details of the MPC protocols are provided in Appendix A.3. The overall algorithm for COPML is presented in Appendix A.5.

4 Convergence and Privacy Guarantees

Consider the cost function in (1) with the quantized dataset, and denote $\mathbf{w}^*$ as the optimal model parameters that minimize (1). In this subsection, we prove that COPML guarantees convergence to the optimal model parameters (i.e., $\mathbf{w}^*$) while maintaining the privacy of the dataset against colluding clients. This result is stated in the following theorem.

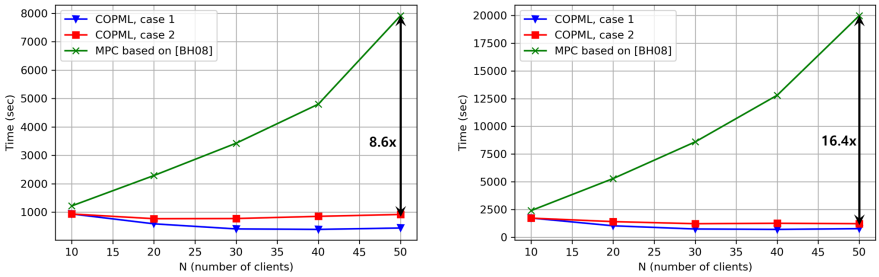
Theorem 1. *For training a logistic regression model in a distributed system with N clients using the quantized dataset $\mathbf{X} = [\mathbf{X}_1^\top, \ldots, \mathbf{X}_N^\top]^\top$, initial model parameters $\mathbf{w}^{(0)}$, and constant step size $\eta \le 1/L$ (where $L = \frac{1}{4}\|\mathbf{X}\|_2^2$), COPML guarantees convergence,*

$$\mathbb{E}\big[C\big(\frac{1}{J}\sum_{t=0}^{J}\mathbf{w}^{(t)}\big)\big] - C(\mathbf{w}^*) \le \frac{\|\,\mathbf{w}^{(0)} - \mathbf{w}^*\,\|^2}{2\eta J} + \eta\sigma^2 \tag{12}$$

in J iterations, for any $N \ge (2r+1)(K+T-1)+1$, where r is the degree of the polynomial in (5) *and σ^2 is the variance of the quantization error of the secure truncation protocol.*

Proof. The proof of Theorem 1 is presented in Appendix A.2. □

As for the privacy guarantees, COPML protects the statistical privacy of the individual dataset of each client against up to T colluding adversarial clients, even if the adversaries have unbounded computational power. The privacy protection of COPML follows from the fact that all building blocks of the algorithm guarantees either (strong) information-theoretic privacy or statistical privacy of the individual datasets against any collusions between up to T clients. Information-theoretic privacy of Lagrange coding against T colluding clients follows from [40]. Moreover, encoding, decoding, and model update operations are carried out in a secure MPC protocol that protects the



(a) CIFAR-10 (for accuracy 80.45%) (b) GISETTE (for accuracy 97.50%)

Figure 3: Performance gain of COPML over the MPC baseline ([BH08] from [3]). The plot shows the total training time for different number of clients N with 50 iterations.

information-theoretic privacy of the corresponding computations against T colluding clients [4, 3, 12]. Finally, the (statistical) privacy guarantees of the truncation protocol follows from [6].

Remark 1. (Privacy-parallelization trade-off) Theorem 1 reveals an important trade-off between privacy and parallelization in COPML. Parameter K reflects the amount of parallelization. In particular, the size of the encoded matrix at each client is equal to $(1/K)^{th}$ of the size of $\mathbf{X}$. Since each client computes the gradient over the encoded dataset, the computation load at each client is proportional to processing $(1/K)^{th}$ of the entire dataset. As K increases, the computation load at each client decreases. Parameter T reflects the privacy threshold of COPML. In a distributed system with N clients, COPML can achieve any K and T as long as $N \geq (2r+1)(K+T-1)+1$. Moreover, as the number of clients N increases, parallelization (K) and privacy (T) thresholds of COPML can also increase linearly, providing a scalable solution. The motivation behind the encoding process is to distribute the load of the computationally-intensive gradient evaluations across multiple clients (enabling parallelization), and to protect the privacy of the dataset.

Remark 2. Theorem 1 also holds for the simpler linear regression problem.

5 Experiments

We demonstrate the performance of COPML compared to conventional MPC baselines by examining two properties, accuracy and performance gain, in terms of the training time on the Amazon EC2 Cloud Platform.

5.1 Experiment setup

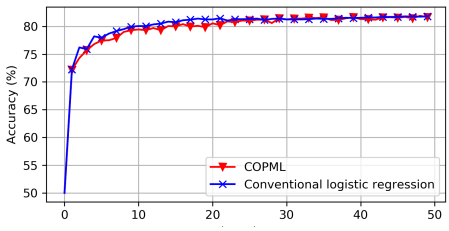
Setup. We train a logistic regression model for binary image classification on the CIFAR-10 [23] and GISETTE [18] datasets, whose size is $(m, d) = (9019, 3073)$ and $(6000, 5000)$, respectively. The dataset is distributed evenly across the clients. The clients initially secret share their individual datasets with the other clients.[5] Computations are carried out on Amazon EC2 `m3.xlarge` machine instances. We run the experiments in a WAN setting with an average bandwidth of $40Mbps$. Communication between clients is implemented using the `MPI4Py` [11] interface on `Python`.

Implemented schemes. We implement four schemes for performance evaluation. For COPML, we consider two set of key parameters (K, T) to investigate the trade-off between parallelization and privacy. For the baselines, we apply two conventional MPC protocols (based on [4] and [3]) to our multi-client problem setting.[6]

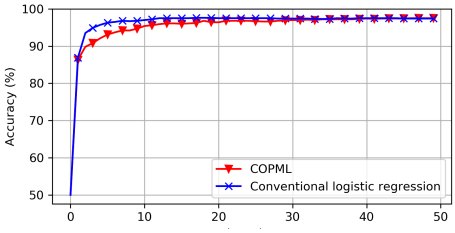
1. **COPML.** In COPML, MPC is utilized to enable secure encoding and decoding for Lagrange coding. The gradient computations are then carried out using the Lagrange encoded data. We determine T (privacy threshold) and K (amount of parallelization) in COPML as follows. Initially, we have from Theorem 1 that these parameters must satisfy $N \geq (2r+1)(K+T-1)+1$ for our framework. Next, we have considered both $r=1$ and $r=3$ for the degree of the polynomial approximation of the sigmoid function and observed that the degree one approximation achieves good accuracy, as we demonstrate later. Given our choice of $r=1$, we then consider two setups:

[5] This can be done offline as it is an identical one-time operation for both MPC baselines and COPML.

[6] As described in the Section 1, there is no prior work at our scale (beyond 3-4 parties), hence we implement two baselines based on well-known MPC protocols which are also the first implementations at our scale.



(a) CIFAR-10 dataset for binary classification between *plain* and *car* images (using 9019 samples for the training set and 2000 samples for the test set).



(b) GISETTE dataset for binary classification between digits 4 and 9 (using 6000 samples for the training set and 1000 samples for the test set).

Figure 4: Comparison of the accuracy of COPML (demonstrated for Case 2 and $N = 50$ clients) vs conventional logistic regression that uses the sigmoid function without quantization.

Case 1: *(Maximum parallelization gain)* Allocate all resources to parallelization (fastest training), by letting $K = \lfloor \frac{N-1}{3} \rfloor$ and $T = 1$,

Case 2: *(Equal parallelization and privacy gain)* Split resources almost equally between parallelization and privacy, i.e., $T = \lfloor \frac{N-3}{6} \rfloor, K = \lfloor \frac{N+2}{3} \rfloor - T$.

2. **Baseline protocols.** We implement two conventional MPC protocols (based on [4] and [3]). In a naive implementation of these protocols, each client would secret share its local dataset with the entire set of clients, and the gradient computations would be performed over the secret shared data whose size is as large as the entire dataset, which leads to a significant computational overhead. For a fair comparison with COPML, we speed up the baseline protocols by partitioning the clients into three groups, and assigning each group one third of the entire dataset. Hence, the total amount of data processed at each client is equal to one third of the size of the entire dataset, which significantly reduces the total training time while providing a privacy threshold of $T = \lfloor \frac{N-3}{6} \rfloor$, which is the same privacy threshold as Case 2 of COPML. The details of these implementations are presented in Appendix A.4.

In all schemes, we apply the MPC truncation protocol from Section 3 to carry out the multiplication with $\frac{\eta}{m}$ during model updates, by choosing $(k_1, k_2) = (21, 24)$ and $(22, 24)$ for the CIFAR-10 and GISETTE datasets, respectively.

5.2 Performance evaluation

Training time. In the first set of experiments, we measure the training time. Our results are demonstrated in Figure 3, which shows the comparison of COPML with the protocol from [3], as we have found it to be the faster of the two baselines. Figures 3(a) and 3(b) demonstrate that COPML provides substantial speedup over the MPC baseline, in particular, up to $8.6\times$ and $16.4\times$ with the CIFAR-10 and GISETTE datasets, respectively, while providing the same privacy threshold T. We observe that a higher amount of speedup is achieved as the dimension of the dataset becomes larger (CIFAR-10 vs. GISETTE datasets), suggesting COPML to be well-suited for data-intensive distributed training tasks where parallelization is essential.

Table 1: Breakdown of the running time with $N = 50$ clients.

Protocol	Comp. time (s)	Comm. time (s)	Enc/Dec time (s)	Total run time (s)
MPC using [BGW88]	918	21142	324	22384
MPC using [BH08]	914	6812	189	7915
COPML (Case 1)	141	284	15	440
COPML (Case 2)	240	654	22	916

To further investigate the gain of COPML, in Table 1 we present the breakdown of the total running time with the CIFAR-10 dataset for $N = 50$ clients. We observe that COPML provides $K/3$ times speedup for the computation time of matrix multiplication in (7), which is given in the first column. This is due to the fact that, in the baseline protocols, the size of the data processed at each client is one third of the entire dataset, while in COPML it is $(1/K)^{th}$ of the entire dataset. This reduces the computational overhead of each client while computing matrix multiplications. Moreover, COPML provides significant improvement in the communication, encoding, and decoding time. This is because the two baseline protocols require intensive communication and computation to carry out a degree reduction step for secure multiplication (encoding and decoding for additional secret shares), which is detailed in Appendix A.3.

Table 2: Complexity summary of COPML.

Communication	Computation	Encoding
$O(\frac{mdN}{K} + dNJ)$	$O(\frac{md^2}{K})$	$O(\frac{mdN(K+T)}{K} + dN(K+T)J)$

In contrast, COPML only requires secure addition and multiplication-by-a-constant operations for encoding and decoding. These operations require no communication. In addition, the communication, encoding, and decoding overheads of each client are also reduced due to the fact that the size of the data processed at each client is only $(1/K)^{th}$ of the entire dataset.

Accuracy. We finally examine the accuracy of COPML. Figures 4(a) and 4(b) demonstrate that COPML with degree one polynomial approximation provides comparable test accuracy to conventional logistic regression. For the CIFAR-10 dataset in Figure 4(a), the accuracy of COPML and conventional logistic regression are 80.45% and 81.75%, respectively, in 50 iterations. For the GISETTE dataset in Figure 4(b), the accuracy of COPML and conventional logistic regression have the same value of 97.5% in 50 iterations. Hence, COPML has comparable accuracy to conventional logistic regression while also being privacy preserving.

5.3 Complexity Analysis

In this section, we analyze the asymptotic complexity of each client in COPML with respect to the number of clients N, model dimension d, number of data points m, parallelization parameter K, privacy parameter T, and total number of iterations J. Client i's communication cost can be broken to three parts: 1) sending the secret shares $[\widetilde{\mathbf{X}}_j]_i = [u(\alpha_j)]_i$ in (3) to client $j \in [N]$, 2) sending the secret shares $[\widetilde{\mathbf{w}}_j^{(t)}]_i = [v(\alpha_j)]_i$ in (4) to client $j \in [N]$ for $t \in \{0, \ldots, J-1\}$, and 3) sending the secret share of local computation $[f(\widetilde{\mathbf{X}}_i, \widetilde{\mathbf{w}}_i^{(t)})]_j$ in (7) to client $j \in [N]$ for $t \in \{0, \ldots, J-1\}$. The communication cost of the three parts are $O(\frac{mdN}{K})$, $O(dNJ)$, and $O(dNJ)$, respectively. Therefore, the overall communication cost of each client is $O(\frac{mdN}{K} + dNJ)$. Client i's computation cost of encoding can be broken into two parts, encoding the dataset by using (3) and encoding the model by using (4). The encoded dataset $[\widetilde{\mathbf{X}}_j]_i = [u(\alpha_j)]_i$ from (3) is a weighted sum of $K+T$ matrices where each matrix belongs to $\mathbb{F}_p^{\frac{m}{K} \times d}$. As there are N encoded dataset and each encoded dataset requires a computation cost of $O(\frac{md(K+T)}{K})$, the computation cost of encoding the dataset is $O(\frac{mdN(K+T)}{K})$ in total. Similarly, computation cost of encoding $[\widetilde{\mathbf{w}}_j^{(t)}]_i = [v(\alpha_j)]_i$ from (4) is $O(dN(K+T)J)$. Computation cost of client i to compute $\widetilde{\mathbf{X}}_i^\top \widetilde{\mathbf{X}}_i$, the dominant part of local computation $f(\widetilde{\mathbf{X}}_i, \widetilde{\mathbf{w}}_i^{(t)})$ in (7), is $O(\frac{md^2}{K})$. We summarize the asymptotic complexity of each client in Table 2.

When we set $N = 3(K+T-1)+1$ and $K = O(N)$ (Case 2), increasing N has two major impacts on the training time: 1) reducing the computation per client by choosing a larger K, 2) increasing the encoding time. In this case, as m is typically much larger than other parameters, dominate terms in communication, computation, and encoding cost are $O(md)$, $O(md^2/N)$ and $O(mdN)$, respectively. For small datasets, i.e., when the computation load at each worker is very small, the gain from increasing the number of workers beyond a certain point may be minimal and system may saturate, as encoding may dominate the computation. This is the reason that a higher amount of speedup of training time is achieved as the dimension of the dataset becomes larger.

6 Conclusions

We considered a collaborative learning scenario in which multiple data-owners jointly train a logistic regression model without revealing their individual datasets to the other parties. To the best of our knowledge, even for the simple logistic regression, COPML is the first fully-decentralized training framework to scale beyond 3-4 parties while achieving information-theoretic privacy. Extending COPML to more complicated (deeper) models is a very interesting future direction. An MPC-friendly (i.e., polynomial) activation function is proposed in [28] which approximates the softmax and shows that the accuracy of the resulting models is very close to those trained using the original functions. We expect to achieve a similar performance gain even in those setups, since COPML can similarly be leveraged to efficiently parallelize the MPC computations.

Broader Impact

Our framework has the societal benefit of protecting user privacy in collaborative machine learning applications, where multiple data-owners can jointly train machine learning models without revealing information about their individual datasets to the other parties, even if some parties collude with each other. Collaboration can significantly improve the accuracy of trained machine learning models, compared to training over individual datasets only. This is especially important in applications where data labelling is costly and can take a long time, such as data collected and labeled in medical fields. For instance, by using our framework, multiple medical institutions can collaborate to train a logistic regression model jointly, without revealing the privacy of their datasets to the other parties, which may contain sensitive patient healthcare records or genetic information. Our framework can scale to a significantly larger number of users compared to the benchmark protocols, and can be applied to any field in which the datasets contain sensitive information, such as healthcare records, financial transactions, or geolocation data. In such applications, protecting the privacy of sensitive information is critical and failure to do so can result in serious societal, ethical, and legal consequences. Our framework can provide both application developers and users with positive societal consequences, application developers can provide better user experience with better models as the volume and diversity of data will be increased greatly, and at the same time, users will have their sensitive information kept private. Another benefit of our framework is that it provides strong privacy guarantees that is independent from the computational power of the adversaries. Therefore, our framework keeps the sensitive user information safe even if adversaries gain quantum computing capabilities in the future.

A potential limitation of our framework is that our current training framework is bound to polynomial operations. In order to compute functions that are not polynomials, such as the sigmoid function, we utilize a polynomial approximation. This can pose a challenge in the future for applying our framework to deep neural network models, as the approximation error may add up at each layer. In such scenarios, one may need to develop additional techniques to better handle the non-linearities and approximation errors.

Acknowledgement

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053, ARO award W911NF1810400, NSF grants CCF-1703575 and CCF-1763673, ONR Award No. N00014-16-1-2189, and research gifts from Intel and Facebook. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

[2] Zuzana Beerliová-Trubíniová. *Efficient multi-party computation with information-theoretic security*. PhD thesis, ETH Zurich, 2008.

[3] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In *Theory of Cryptography Conference*, pages 213–230. Springer, 2008.

[4] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM Symp. on Th. of Comp.*, pages 1–10, 1988.

[5] J. Brinkhuis and V. Tikhomirov. *Optimization: Insights and Applications*. Princeton Series in Applied Mathematics. Princeton University Press, 2011.

[6] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *International Conference on Financial Cryptography and Data Security*, pages 35–50. Springer, 2010.

[7] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Adv. in Neural Inf. Proc. Sys.*, pages 289–296, 2009.

[8] Valerie Chen, Valerio Pastro, and Mariana Raykova. Secure computation for machine learning with SPDZ. *arXiv:1901.00329*, 2019.

[9] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography Conference*, pages 342–362. Springer, 2005.

[10] Morten Dahl, Jason Mancuso, Yann Dupis, Ben Decoste, Morgan Giraud, Ian Livingstone, Justin Patriquin, and Gavin Uhma. Private machine learning in TensorFlow using secure computation. *arXiv:1810.08130*, 2018.

[11] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. MPI for Python. *Journal of Parallel and Distributed Comp.*, 65(9):1108–1115, 2005.

[12] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*, pages 572–590. Springer, 2007.

[13] Matthew Franklin and Moti Yung. Communication complexity of secure computation. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 699–710. ACM, 1992.

[14] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Tech.*, 2017(4):345–364, 2017.

[15] Craig Gentry and Dan Boneh. *A fully homomorphic encryption scheme*, volume 20. Stanford University, Stanford, 2009.

[16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Int. Conf. on Machine Learning*, pages 201–210, 2016.

[17] Thore Graepel, Kristin Lauter, and Michael Naehrig. ML confidential: Machine learning on encrypted data. In *Int. Conf. on Information Security and Cryptology*, pages 1–21. Springer, 2012.

[18] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *Advances in Neural Inf. Processing Systems*, pages 545–552. 2005.

[19] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Logistic regression on homomorphic encrypted data at scale. *Annual Conf. on Innovative App. of Artificial Intelligence (IAAI-19)*, 2019.

[20] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. CryptoDL: Deep neural networks over encrypted data. *arXiv:1711.05189*, 2017.

[21] Bargav Jayaraman, Lingxiao Wang, David Evans, and Quanquan Gu. Distributed learning without distress: Privacy-preserving empirical risk minimization. *Adv. in Neur. Inf. Pro. Sys.*, pages 6346–6357, 2018.

[22] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11(4):83, 2018.

[23] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[24] Ping Li, Jin Li, Zhengan Huang, Chong-Zhi Gao, Wen-Bin Chen, and Kai Chen. Privacy-preserving outsourced classification in cloud computing. *Cluster Computing*, pages 1–10, 2017.

[25] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Int. Cryptology Conf.*, pages 36–54. Springer, 2000.

[26] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *Int. Conf. on Learning Representations*, 2018.

[27] Payman Mohassel and Peter Rindal. ABY 3: A mixed protocol framework for machine learning. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 35–52, 2018.

[28] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *38th IEEE Symposium on Security and Privacy*, pages 19–38. IEEE, 2017.

[29] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition, 2014.

[30] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE Symposium on Security and Privacy*, pages 334–348, 2013.

[31] Manas Pathak, Shantanu Rane, and Bhiksha Raj. Multiparty differential privacy via aggregation of locally trained classifiers. In *Advances in Neural Inf. Processing Systems*, pages 1876–1884, 2010.

[32] Arun Rajkumar and Shivani Agarwal. A differentially private stochastic gradient descent algorithm for multiparty classification. In *Int. Conf. on Artificial Intelligence and Statistics (AISTATS'12)*, volume 22, pages 933–941, La Palma, Canary Islands, Apr 2012.

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

[34] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1310–1321, 2015.

[35] Jinhyun So, Basak Guler, and A Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *arXiv preprint arXiv:2002.04156*, 2020.

[36] Jinhyun So, Basak Guler, A Salman Avestimehr, and Payman Mohassel. Codedprivateml: A fast and privacy-preserving framework for distributed machine learning. *arXiv preprint arXiv:1902.00641*, 2019.

[37] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, 2019.

[38] Q. Wang, M. Du, X. Chen, Y. Chen, P. Zhou, X. Chen, and X. Huang. Privacy-preserving collaborative model learning: The case of word vector training. *IEEE Trans. on Knowledge and Data Engineering*, 30(12):2381–2393, Dec 2018.

[39] Andrew C Yao. Protocols for secure computations. In *IEEE Symp. on Foundations of Computer Science*, pages 160–164, 1982.

[40] Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and A Salman Avestimehr. Lagrange coded computing: Optimal design for resiliency, security and privacy. In *Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, 2019.

[41] Jiawei Yuan and Shucheng Yu. Privacy preserving back-propagation neural network learning made practical with cloud computing. *IEEE Trans. on Parallel and Dist. Sys.*, 25(1):212–221, 2014.

A Supplementary Materials

A.1 Details of the Quantization Phase

For quantizing its dataset $\mathbf{X}_j$, client $j \in [N]$ employs a scalar quantization function $\phi\left(Round(2^{l_x} \cdot \mathbf{X}_j)\right)$, where the rounding operation

$$Round(x) = \begin{cases} \lfloor x \rfloor & \text{if} \quad x - \lfloor x \rfloor < 0.5 \\ \lfloor x \rfloor + 1 & \text{otherwise} \end{cases} \tag{13}$$

is applied element-wise to the elements x of matrix $\mathbf{X}_j$ and l_x is an integer parameter to control the quantization loss. $\lfloor x \rfloor$ is the largest integer less than or equal to x, and function $\phi : \mathbb{Z} \to \mathbb{F}_p$ is a mapping defined to represent a negative integer in the finite field by using two's complement representation,

$$\phi(x) = \begin{cases} x & \text{if } x \geq 0 \\ p + x & \text{if } x < 0 \end{cases} \tag{14}$$

To avoid a wrap-around which may lead to an overflow error, prime p should be large enough, $p \geq 2^{l_x+1} \max\{|x|\} + 1$. Its value also depends on the bitwidth of the machine as well as the dimension of the dataset. For example, in a 64-bit implementation with the CIFAR-10 dataset whose dimension is $d = 3072$, we select $p = 2^{26} - 5$, which is the largest prime needed to avoid an overflow on intermediate multiplications. In particular, in order to speed up the running time of matrix-matrix multiplication, we do a modular operation after the inner product of vectors instead of doing a modular operation per product of each element. To avoid an overflow on this, p should be smaller than a threshold given by $d(p-1)^2 \leq 2^{64} - 1$. For ease of exposition, throughout the paper, $\mathbf{X} = [\mathbf{X}_1^\top, \ldots, \mathbf{X}_N^\top]^\top$ refers to the quantized dataset.

A.2 Proof of Theorem 1

First, we show that the minimum number of clients needed for our decoding operation to be successful, i.e., the recovery threshold of COPML, is equal to $(2r+1)(K+T-1)+1$. To do so, we demonstrate in the following that the decoding process will be successful as long as $N \geq (2r+1)(K+T-1)+1$. As described in Section 3, given the polynomial approximation of the sigmoid function in (5), the degree of $h(z)$ in (8) is at most $(2r+1)(K+T-1)$. The decoding process uses the computations from the clients as evaluation points $h(\alpha_i)$ to interpolate the polynomial $h(z)$. If at least $deg(h(z))+1$ evaluation results of $h(\alpha_i)$ are available, then, all of the coefficients of $h(z)$ can be evaluated. After $h(z)$ is recovered, the sub-gradient $\mathbf{X}_i^\top \hat{g}(\mathbf{X}_i \times \mathbf{w}^{(t)})$ can be decoded by computing $h(\beta_i)$ for $i \in [K]$, from which the gradient $\mathbf{X}^\top \hat{g}(\mathbf{X} \times \mathbf{w}^{(t)})$ from (11) can be computed. Hence, the recovery threshold of COPML is $(2r+1)(K+T-1)+1$, as long as $N \geq (2r+1)(K+T-1)+1$, the protocol can correctly decode the gradient using the local evaluations of the clients, and the decoding process will be successful. Since the decoding operations are performed using a secure MPC protocol, throughout the decoding process, the clients only learn a secret share of the gradient and not its actual value. Next, we consider the update equation in (6) and prove its convergence to $\mathbf{w}^*$. As described in Section 3, after decoding the gradient, the clients carry out a secure truncation protocol to multiply $\mathbf{X}^\top(\hat{g}(\mathbf{X} \times \mathbf{w}^{(t)}) - \mathbf{y})$ with parameter $\frac{\eta}{m}$ to update the model as in (6). The update equation from (6) can then be represented by

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta\big(\frac{1}{m}\mathbf{X}^\top(\hat{g}(\mathbf{X} \times \mathbf{w}^{(t)}) - \mathbf{y}) + \mathbf{n}^{(t)}\big). \tag{15}$$

$$= \mathbf{w}^{(t)} - \eta\mathbf{p}^{(t)} \tag{16}$$

where $\mathbf{n}^{(t)}$ represents the quantization noise introduced by the secure multi-party truncation protocol [6], and $\mathbf{p}^{(t)} \triangleq \frac{1}{m}\mathbf{X}^\top(\hat{g}(\mathbf{X} \times \mathbf{w}^{(t)}) - \mathbf{y}) + \mathbf{n}^{(t)}$. From [6], $\mathbf{n}^{(t)}$ has zero mean and bounded variance, i.e., $\mathbb{E}_{\mathbf{n}^{(t)}}[\mathbf{n}^{(t)}] = 0$ and $\mathbb{E}_{\mathbf{n}^{(t)}}\left[\|\mathbf{n}^{(t)}\|_2^2\right] \leq \frac{d2^{2(k_1-1)}}{m^2} \triangleq \sigma^2$ where $\|\cdot\|_2$ is the l_2 norm and k_1 is the truncation parameter described in Section 3.

Next, we show that $\mathbf{p}^{(t)}$ is an unbiased estimator of the true gradient, $\nabla C(\mathbf{w}^{(t)}) = \frac{1}{m}\mathbf{X}^\top(g(\mathbf{X} \times \mathbf{w}^{(t)}) - \mathbf{y})$, and its variance is bounded by σ^2 with sufficiently large r. From $\mathbb{E}_{\mathbf{n}^{(t)}}[\mathbf{n}^{(t)}] = 0$, we obtain

$$\mathbb{E}_{\mathbf{n}^{(t)}}[\mathbf{p}^{(t)}] - \nabla C(\mathbf{w}^{(t)}) = \frac{1}{m}\mathbf{X}^\top\left(\hat{g}(\mathbf{X} \times \mathbf{w}^{(t)}) - g(\mathbf{X} \times \mathbf{w}^{(t)})\right). \tag{17}$$

From the Weierstrass approximation theorem [5], for any $\epsilon > 0$, there exists a polynomial that approximates the sigmoid arbitrarily well, i.e., $|\hat{g}(x) - g(x)| \leq \epsilon$ for all x in the constrained interval. Hence, as there exists a polynomial making the norm of (17) arbitrarily small, $\mathbb{E}_{\mathbf{n}^{(\mathrm{t})}}[\mathbf{p}^{(t)}] = \nabla C(\mathbf{w}^{(t)})$ and $\mathbb{E}_{\mathbf{n}^{(\mathrm{t})}}\left[\|\mathbf{p}^{(t)} - \mathbb{E}_{\mathbf{n}^{(\mathrm{t})}}[\mathbf{p}^{(t)}]\|_2^2\right] = \mathbb{E}_{\mathbf{n}^{(\mathrm{t})}}\left[\|\mathbf{n}^{(t)}\|_2^2\right] \leq \sigma^2$.

Next, we consider the update equation in (16) and prove its convergence to $\mathbf{w}^*$. From the L-Lipschitz continuity of $\nabla C(\mathbf{w})$ (Theorem 2.1.5 of [29]), we have

$$\begin{aligned} C(\mathbf{w}^{(t+1)}) &\leq C(\mathbf{w}^{(t)}) + \langle \nabla C(\mathbf{w}^{(t)}), \mathbf{w}^{(t+1)} - \mathbf{w}^{(t)} \rangle + \frac{L}{2} \| \mathbf{w}^{(t+1)} - \mathbf{w}^{(t)} \|^2 \\ &\leq C(\mathbf{w}^{(t)}) - \eta \langle \nabla \mathrm{C}(\mathbf{w}^{(t)}), \mathbf{p}^{(t)} \rangle + \frac{L\eta^2}{2} \|\mathbf{p}^{(t)}\|^2, \end{aligned} \tag{18}$$

where $\langle \cdot, \cdot \rangle$ is the inner product. For a cross entropy loss $C(\mathbf{w})$, the Lipschitz constant L is equal to the largest eigenvalue of the Hessian $\nabla^2 C(\mathbf{w})$ for all $\mathbf{w}$, and is given by $L = \frac{1}{4}\|\mathbf{X}\|_2^2$. By taking the expectation with respect to the quantization noise $\mathbf{n}^{(\mathrm{t})}$ on both sides in (18), we have

$$\mathbb{E}_{\mathbf{n}^{(\mathrm{t})}}\left[C(\mathbf{w}^{(t+1)})\right] \leq C(\mathbf{w}^{(t)}) - \eta \| \nabla \mathrm{C}(\mathbf{w}^{(t)}) \|^2 + \frac{L\eta^2}{2} \left(\| \nabla \mathrm{C}(\mathbf{w}^{(t)}) \|^2 + \sigma^2 \right) \tag{19}$$

$$\leq C(\mathbf{w}^{(t)}) - \eta \left(1 - \frac{L\eta}{2}\right) \| \nabla \mathrm{C}(\mathbf{w}^{(t)}) \|^2 + \frac{L\eta^2\sigma^2}{2}$$

$$\leq C(\mathbf{w}^{(t)}) - \frac{\eta}{2} \| \nabla \mathrm{C}(\mathbf{w}^{(t)}) \|^2 + \frac{\eta\sigma^2}{2} \tag{20}$$

$$\leq C(\mathbf{w}^*) + \langle \nabla \mathrm{C}(\mathbf{w}^{(t)}), \mathbf{w}^{(t)} - \mathbf{w}^* \rangle - \frac{\eta}{2} \| \nabla \mathrm{C}(\mathbf{w}^{(t)}) \|^2 + \frac{\eta\sigma^2}{2} \tag{21}$$

$$\leq C(\mathbf{w}^*) + \langle \mathbb{E}_{\mathbf{n}^{(\mathrm{t})}}[\mathbf{p}^{(t)}], \mathbf{w}^{(t)} - \mathbf{w}^* \rangle - \frac{\eta}{2} \mathbb{E}_{\mathbf{n}^{(\mathrm{t})}} \|\mathbf{p}^{(t)})\|^2 + \eta\sigma^2 \tag{22}$$

$$= C(\mathbf{w}^*) + \eta\sigma^2 + \mathbb{E}_{\mathbf{n}^{(\mathrm{t})}} \left[\langle \mathbf{p}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \rangle - \frac{\eta}{2} \|\mathbf{p}^{(t)})\|^2 \right]$$

$$= C(\mathbf{w}^*) + \eta\sigma^2 + \frac{1}{2\eta} \left(\| \mathbf{w}^{(t)} - \mathbf{w}^* \|^2 - \mathbb{E}_{\mathbf{n}^{(\mathrm{t})}} \| \mathbf{w}^{(t+1)} - \mathbf{w}^*) \|^2 \right) \tag{23}$$

where (19) and (22) hold since $\mathbb{E}_{\mathbf{n}^{(\mathrm{t})}}[\mathbf{p}^{(t)}] = \nabla C(\mathbf{w}^{(t)})$ and $\mathbb{E}_{\mathbf{n}^{(\mathrm{t})}}\left[\|\mathbf{p}^{(t)} - \nabla C(\mathbf{w}^{(t)})\|_2^2\right] \leq \sigma^2$, (20) follows from $L\eta \leq 1$, (21) follows from the convexity of C, and (23) follows from $\mathbf{p}^{(t)} = -\frac{1}{\eta}(\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)})$.

By taking the expectation on both sides in (23) with respect to the joint distribution of all random variables $\mathbf{n}^{(0)}, \ldots, \mathbf{n}^{(J-1)}$ where J denotes the total number of iterations, we have

$$\mathbb{E}\left[C(\mathbf{w}^{(t+1)})\right] - C(\mathbf{w}^*) \leq \frac{1}{2\eta} \left(\mathbb{E} \| \mathbf{w}^{(t)} - \mathbf{w}^* \|^2 - \mathbb{E} \| \mathbf{w}^{(t+1)} - \mathbf{w}^*) \|^2 \right) + \eta\sigma^2. \tag{24}$$

Summing both sides of the inequality in (24) for $t = 0, \ldots, J-1$, we find that,

$$\sum_{t=0}^{J-1} \left(\mathbb{E}\left[C(\mathbf{w}^{(t+1)})\right] - C(\mathbf{w}^*) \right) \leq \frac{\| \mathbf{w}^{(0)} - \mathbf{w}^* \|^2}{2\eta} + J\eta\sigma^2.$$

Finally, since C is convex, we observe that,

$$\begin{aligned} \mathbb{E}\left[C\left(\frac{1}{J}\sum_{t=0}^{J} \mathbf{w}^{(t)}\right)\right] - C(\mathbf{w}^*) &\leq \frac{1}{J} \sum_{t=0}^{J-1} \left(\mathbb{E}\left[C(\mathbf{w}^{(t+1)})\right] - C(\mathbf{w}^*) \right) \\ &\leq \frac{\| \mathbf{w}^{(0)} - \mathbf{w}^* \|^2}{2\eta J} + \eta\sigma^2 \end{aligned}$$

which completes the proof of convergence.

A.3 Details of the Multi-Party Computation (MPC) Implementation

We consider two well-known MPC protocols, the notable BGW protocol from [4], and the more recent, efficient MPC protocol from [3, 12]. Both protocols allow the computation of any polynomial

function in a privacy-preserving manner by untrusted parties. Computations are carried out over the secret shares, and at the end, parties only learn a secret share of the actual result. Any collusions between up to $T = \lfloor \frac{N-1}{2} \rfloor$ out of N parties do not reveal information (in an information-theoretic sense) about the input variables. The latter protocol is more efficient in terms of the communication cost between the parties, which scales linearly with respect to the number of parties, whereas for the former protocol this cost is quadratic. As a trade-off, it requires a considerable amount of offline computations and higher storage cost for creating and secret sharing the random variables used in the protocol.

For creating secret shares, we utilize Shamir's T-out-of-N secret sharing [33]. This scheme embeds a secret a in a degree T polynomial $h(\xi) = a + \xi v_1, \ldots, \xi^T v_T$ where v_i, $i \in [T]$ are uniformly random variables. Client $i \in [N]$ then receives a secret share of a, denoted by $h(i) = [a]_i$. This keeps a private against any collusions between up to any T parties. The specific computations are then carried out as follows.

Addition. In order to perform a secure addition $a + b$, clients locally add their secret shares $[a]_i + [b]_i$. The resulting value is a secret share of the original summation $a + b$. This step requires no communication.

Multiplication-by-a-constant. For performing a secure multiplication ac where c is a publicly-known constant, clients locally multiply their secret share $[a]_i$ with c. The resulting value is a secret share of the desired multiplication ac. This step requires no communication.

Multiplication. For performing a secure multiplication ab, the two protocols differ in their execution. In the BGW protocol, each client initially multiplies its secret shares $[a]_i$, $[b]_i$ locally to obtain $[a]_i[b]_i$. The clients will then be holding a secret share of ab, however, the corresponding polynomial now has degree $2T$. This may in turn cause the degree of the polynomial to increase excessively as more multiplication operations are evaluated. To alleviate this problem, in the next phase, clients carry out a degree reduction step to create new shares corresponding to a polynomial of degree T. The communication overhead of this protocol is $O(N^2)$.

The protocol from [3], on the other hand, leverages offline computations to speed up the communication phase. In particular, a random variable ρ is created offline and secret shared with the clients twice using two random polynomials with degrees T and $2T$, respectively. The secret shares corresponding to the degree T polynomial are denoted by $[\rho]_{T,i}$, whereas the secret shares for the degree $2T$ polynomial are denoted by $[\rho]_{2T,i}$ for clients $i \in [N]$. In the online phase, client $i \in [N]$ locally computes the multiplication $[a]_i[b]_i$, after which each client will be holding a secret share of the multiplication ab. The resulting polynomial has degree $2T$. Then, each client locally computes $[a]_i[b]_i - [\rho]_{2T,i}$, which corresponds to a secret share of $ab - \rho$ embedded in a degree $2T$ polynomial. Clients then broadcast their individual computations to others, after which each client computes $ab - \rho$. Note that the privacy of the computation ab is still protected since clients do not know the actual value of ab, but instead its masked version $ab - \rho$. Then, each client locally computes $ab - \rho + [\rho]_{T,i}$. As a result, variable ρ cancels out, and clients obtain a secret share of the multiplication ab embedded in a degree T polynomial. This protocol requires only $O(N)$ broadcasts and therefore is more efficient than the previous algorithm. On the other hand, it requires an offline computation phase and higher storage overhead. For the details, we refer to [3, 2].

Remark 3. The secure MPC computations during the encoding, decoding, and model update phases of COPML only use addition and multiplication-by-a-constant operations, instead of the expensive multiplication operation, as $\{\alpha_i\}_{i\in[N]}$ and $\{\beta_k\}_{k\in[K+T]}$ are publicly known constants for all clients.

A.4 Details of the Optimized Baseline Protocols

In a naive implementation of our multi-client problem setting, both baseline protocols would utilize Shamir's secret sharing scheme where the quantized dataset $\mathbf{X} = [\mathbf{X}_1^\top, \ldots, \mathbf{X}_N^\top]^\top$ is secret shared with N clients. To do so, both baselines would follow the same secret sharing process as in COPML, where client $j \in [N]$ creates a degree T random polynomial $h_j(z) = \mathbf{X}_j + z\mathbf{R}_{j1} + \ldots + z^T\mathbf{R}_{jT}$ where $\mathbf{R}_{ji}$ for $i \in [T]$ are i.i.d. uniformly distributed random matrices while selecting $T = \lfloor \frac{N-1}{2} \rfloor$. By selecting N distinct evaluation points $\lambda_1, \ldots, \lambda_N$ from $\mathbb{F}_p$, client j would generate and send $[\mathbf{X}_j]_i = h_j(\lambda_i)$ to client $i \in [N]$. As a result, client $i \in [N]$ would be assigned a secret share of the entire dataset $\mathbf{X}$, i.e, $[\mathbf{X}]_i = \left[[\mathbf{X}_1]_i^\top, \ldots, [\mathbf{X}_N]_i^\top\right]^\top$. Client i would also obtain a secret share of the

labels, $[\mathbf{y}]_i$, and a secret share of the initial model, $[\mathbf{w}^{(0)}]_i$, where $\mathbf{y} = [\mathbf{y}_1^\top, \ldots, \mathbf{y}_N^\top]^\top$ and $\mathbf{w}^{(0)}$ is a randomly initialized model. Then, the clients would compute the gradient and update the model from (7) within a secure MPC protocol. This guarantees privacy against $\lfloor \frac{N-1}{2} \rfloor$ colluding workers, but requires a computation load at each worker that is as large as processing the whole dataset at a single worker, leading to slow training.

Hence, in order to provide a fair comparison with COPML, we optimize (speed up) the baseline protocols by partitioning the clients into subgroups of size $2T + 1$. Clients communicate a secret share of their own datasets with the other clients in the same subgroup, instead of secret sharing it with the entire set of clients. Each client in subgroup i receives a secret share of a partitioned dataset $\mathbf{X}_i \in \mathbb{F}_p^{\frac{m}{G} \times d}$ where $\mathbf{X} = [\mathbf{X}_1^\top \cdots \mathbf{X}_G^\top]^\top$ and G is the number of subgroups. In other words, client j in subgroup i obtains a secret share $[\mathbf{X}_i]_j$. Then, subgroup $i \in [G]$ computes the sub-gradient over the partitioned dataset, $\mathbf{X}_i$, within a secure MPC protocol. To provide the same privacy threshold $T = \lfloor \frac{N-3}{6} \rfloor$ as Case 2 of COPML in Section 5, we set $G = 3$. This significantly reduces the total training time of the two baseline protocols (compared to the naive MPC implementation where the computation load at each client would be as high as training centrally), as the total amount of data processed at each client is equal to one third of the size of the entire dataset $\mathbf{X}$.

A.5 Algorithms

The overall procedure of COPML protocol is given in Algorithm 1.

Algorithm 1 COPML

input Dataset $(\mathbf{X}, \mathbf{y}) = ((\mathbf{X}_1, \mathbf{y}_1), \ldots, (\mathbf{X}_N, \mathbf{y}_N))$ distributed over N clients.
output Model parameters $\mathbf{w}^{(J)}$.

for client $j = 1, \ldots, N$ **do**
  Secret share the individual dataset $(\mathbf{X}_j, \mathbf{y}_j)$ with clients $i \in [N]$.
end for
Within a secure MPC protocol, initialize the model $\mathbf{w}^{(0)}$ randomly and secret share with clients $i \in [N]$.
// Client i receives a secret share $[\mathbf{w}^{(0)}]_i$ of $\mathbf{w}^{(0)}$.
Encode the dataset within a secure MPC protocol, using the secret shares $[\mathbf{X}_j]_i$ for $j \in [N]$, $i \in [N]$.
// After this step, client i holds a secret share $[\widetilde{\mathbf{X}}_j]_i$ of each encoded dataset $\widetilde{\mathbf{X}}_j$ for $j \in [N]$.
for client $i = 1, \ldots, N$ **do**
  Gather the secret shares $[\widetilde{\mathbf{X}}_i]_j$ from clients $j \in [N]$.
  Recover the encoded dataset $\widetilde{\mathbf{X}}_i$ from the secret shares $\{[\widetilde{\mathbf{X}}_i]_j\}_{j\in[N]}$.
  // At the end of this step, client i obtains the encoded dataset $\widetilde{\mathbf{X}}_i$.
end for
Compute $\mathbf{X}^T \mathbf{y}$ within a secure MPC protocol using the secret shares $[\mathbf{X}_j]_i$ and $[\mathbf{y}_j]_i$ for $j \in [N]$, $i \in [N]$.
// At the end of this step, client i holds a secret share $[\mathbf{X}^T \mathbf{y}]_i$ of $\mathbf{X}^T \mathbf{y}$.
for iteration $t = 0, \ldots, J-1$ **do**
  Encode the model $\mathbf{w}^{(t)}$ in a secure MPC protocol using the secret shares $[\mathbf{w}^{(t)}]_i$.
  // After this step, client i holds a secret share $[\widetilde{\mathbf{w}}_j^{(t)}]_i$ of the encoded model $\widetilde{\mathbf{w}}_j^{(t)}$ for $j \in [N]$.
  for client $i = 1, \ldots, N$ **do**
    Gather the secret shares $[\widetilde{\mathbf{w}}_i^{(t)}]_j$ from clients $j \in [N]$.
    Recover the encoded model $\widetilde{\mathbf{w}}_i^{(t)}$ from the secret shares $\{[\widetilde{\mathbf{w}}_i^{(t)}]_j\}_{j\in[N]}$.
    // At the end of this step, client i obtains the encoded model $\widetilde{\mathbf{w}}_i^{(t)}$.
    Locally compute $f(\widetilde{\mathbf{X}}_i, \widetilde{\mathbf{w}}_i^{(t)})$ from (7) and secret share the result with clients $j \in [N]$.
    // Client i sends a secret share $[f(\widetilde{\mathbf{X}}_i, \widetilde{\mathbf{w}}_i^{(t)})]_j$ of $f(\widetilde{\mathbf{X}}_i, \widetilde{\mathbf{w}}_i^{(t)})$ to client j.
  end for
  for client $i = 1, \ldots, N$ **do**
    Locally computes $[f(\mathbf{X}_k, \mathbf{w}^{(t)})]_i$ for $k \in [K]$ from (10).
    // After this step, client i knows a secret share $[f(\mathbf{X}_k, \mathbf{w}^{(t)})]_i$ of $f(\mathbf{X}_k, \mathbf{w}^{(t)})$ for $k \in [K]$.
    Locally aggregate the secret shares $\{[f(\mathbf{X}_k, \mathbf{w}^{(t)})]_i\}_{k\in\mathcal{K}}$ to compute $[\mathbf{X}^T \hat{g}(\mathbf{X} \times \mathbf{w}^{(t)})]_i \triangleq \sum_{k\in[K]} [f(\mathbf{X}_k, \mathbf{w}^{(t)})]_i$.
    // At the end of this step, client i now has a secret share $[\mathbf{X}^T \hat{g}(\mathbf{X} \times \mathbf{w}^{(t)})]_i$ of $\mathbf{X}^T \hat{g}(\mathbf{X} \times \mathbf{w}^{(t)}) = \sum_{k\in[K]} f(\mathbf{X}_k, \mathbf{w}^{(t)})$.
    Locally compute $[\mathbf{X}^\top(\hat{g}(\mathbf{X} \times \mathbf{w}^{(t)}) - \mathbf{y})]_i \triangleq [\mathbf{X}^T \hat{g}(\mathbf{X} \times \mathbf{w}^{(t)})]_i - [\mathbf{X}^T \mathbf{y}]_i$.
    // Each client now has a secret share $[\mathbf{X}^\top(\hat{g}(\mathbf{X} \times \mathbf{w}^{(t)}) - \mathbf{y})]_i$ of $\mathbf{X}^\top(\hat{g}(\mathbf{X} \times \mathbf{w}^{(t)}) - \mathbf{y})$.
  end for
  Update the model according to (6) within a secure MPC protocol using the secret shares $[\mathbf{X}^\top(\hat{g}(\mathbf{X} \times \mathbf{w}^{(t)}) - \mathbf{y})]_i$ and $[\mathbf{w}^{(t)}]_i$ for $i \in [N]$, and by carrying out the secure truncation operation.
  // At the end of this step, client i holds a secret share of the updated model $[\mathbf{w}^{(t+1)}]_i$.
  // Secure truncation is carried out jointly as it requires communication between the clients.
end for
for client $j = 1, \ldots, N$ **do**
  Collect the secret shares $[\mathbf{w}^{(J)}]_i$ from clients $i \in [N]$ and recover the final model $\mathbf{w}^{(J)}$.
end for