Breaking the Quadratic Communication Overhead of Secure Multi-Party Neural Network Training

Xingyu Lu and Başak Güler
Department of Electrical and Computer Engineering
University of California, Riverside
xlu065@ucr.edu, bguler@ece.ucr.edu

2023 IEEE International Symposium on Information Theory (ISIT) | 978-1-6654-7554-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISIT54713.2023.10206617

***Abstract*—Privacy-preserving machine learning has achieved exciting breakthroughs for collaboratively training machine learning models under strong information-theoretic privacy guarantees. Despite the recent advances, communication bottleneck still remains as a major challenge against scalability to large neural networks. To address this challenge, in this work we introduce CLOVER, the first multi-party neural network training framework with linear communication complexity, significantly improving over the quadratic state-of-the-art, under strong end-to-end information-theoretic privacy guarantees. CLOVER builds on a novel degree reduction mechanism with linear communication complexity, termed *Double Lagrange Coding*, for coded computing. While providing strong multi-round information-theoretic privacy guarantees, CLOVER achieves equal adversary tolerance, resilience to user dropouts, and model accuracy as the state-of-the-art, while significantly cutting down the communication overhead. In doing so, CLOVER addresses a key technical challenge in collaborative neural network training, paving the way for large-scale privacy-aware deep learning applications.**

I. Introduction

Privacy-preserving collaborative machine learning (PPML) is a popular paradigm for training ML models across multiple data-owners (users), without compromising the privacy of local data [1]–[9]. Lagrange coded computing (LCC) is a recent information-theoretic approach that has led to breakthrough advances in PPML [10]–[12]. At the outset, LCC utilizes a Lagrange interpolation polynomial to *encode* the datasets by injecting randomness and (computational) redundancy across the computations performed by different users. Training is then performed on the encoded datasets, *as if they were performed on the true datasets*. The additional randomness is *reversible*; after multiple training rounds, the final model can be correctly recovered using polynomial interpolation (over the computations performed on the encoded data). LCC provides strong information-theoretic privacy guarantees for the sensitive data as well as resilience against user drop-outs, while achieving an order-of-magnitude speed-up in training compared to state-of-the-art cryptographic baselines [11].

Quadratic communication bottleneck. The major challenge against the scalability of information-theoretic PPML frameworks is their *quadratic communication complexity* in the number of users. This is due to the fact that interpolating a polynomial f of degree $\deg(f)$ requires collecting the computation results from at least $N \geq \deg(f)+1$ users. On the other hand, the polynomial degree grows exponentially with each multiplicative operation (associated with gradient computations), causing a *degree explosion* after only a few training rounds, where the total number of users will no longer be sufficient to recover the final model. To reduce the polynomial degree (without breaching privacy), users then need to carry out an expensive *degree reduction protocol*, which incurs a quadratic communication overhead, preventing scalability to larger networks. As a result, current large-scale PPML applications (beyond 3-4 users) with end-to-end information-theoretic privacy guarantees (users learn no information beyond the final model) can apply only to simpler logistic and linear regression tasks, as opposed to more complex neural network training.

Contributions. To address this challenge, in this work we introduce CLOVER (ColLabOrative priVate nEural network tRaining), the first information-theoretic PPML framework with linear communication complexity, for *neural network training*. The key ingredient of CLOVER is a highly efficient novel degree reduction mechanism for LCC, which reduces the quadratic communication overhead to linear, without compromising privacy. To do so, we separate communication into online (data-dependent) and offline (data-agnostic) components. The former depends on the data, and can only be carried out after training starts, whereas the latter is independent from data (e.g., randomness generation), and can be carried out in advance when network load is low. In doing so, we offload the communication-intensive operations (with quadratic overhead) to the offline phase, by trading off the quadratic (point-to-point) communication with linear (broadcast). Then, in the offline phase, we introduce a novel randomness generation mechanism for LCC, where the total number of variables communicated is inversely proportional to the number of users, achieving a linear (amortized) communication complexity.

In our theoretical analysis, we demonstrate the formal information-theoretic privacy guarantees of CLOVER, as well as the key performance trade-offs in terms of the communication complexity, robustness against user dropouts, and adversary resilience. In a network of N users, we show that CLOVER achieves an $O(N)$ (linear) communication complexity both offline and online (for neural network training), as opposed to the $O(N^2)$ (quadratic) online communication complexity of the state of the art (which, as a result, is limited to simpler logistic/linear regression, as opposed to neural networks) [11], while achieving equal adversary and dropout resilience. Our contributions are summarized as follows:

This research was sponsored in part by the OUSD (R&E)/RT&L under Cooperative Agreement Number W911NF-20-2-0267, NSF CAREER Award CCF-2144927, UCR OASIS Award and UC Regents Faculty Fellowship.

- We propose CLOVER, the first privacy-preserving multi-party neural network training framework with linear communication complexity, under strong end-to-end information-theoretic privacy guarantees.
- We introduce the first degree reduction mechanism for LCC with linear communication complexity, which can open up further research in a broad range of privacy-preserving iterative algorithms.
- We present the formal information-theoretic privacy guarantees for end-to-end neural network training, and show that CLOVER cuts the communication overhead while achieving equal adversary resilience, model accuracy, and robustness to user dropouts as the state-of-the-art.

Related work. Beyond coded computing, there are three complementary approaches to PPML. *Secure Multi-party Computing* is a cryptographic PPML framework that injects (reversible) randomness to sensitive data before training [2]–[9]. Their main challenge is the extensive communication required between the users, which limits scalability beyond $2-4$ users. *Differential Privacy (DP)* protects privacy by injecting (irreversible) noise during training, so that an adversary cannot backtrack an individual's data from the final model [13]–[20]. DP has an inherent accuracy-privacy trade-off; stronger privacy requires a higher noise level. It was shown recently that DP can also be integrated with information-theoretic PPML, to reduce the DP noise in distributed settings [21]–[23]. Though beyond our current scope, we note this as an interesting future direction. *Homomorphic Encryption* performs computations on encrypted data [24]–[34]. Stronger privacy requires larger encrypted data, increasing the computation load per user.

II. Problem Formulation

Multi-party Neural Network Training. We consider a network of N users. User i holds a local dataset represented by a matrix $\mathbf{X}_i$ of size $d \times m_i$, where m_i and d denote the number of data points and features, respectively. The corresponding labels are represented by a binary matrix $\mathbf{Y}_i$ of size $c \times m_i$ for $i \in [N]$, where the k^{th} column is the one-hot label vector for data point $k \in [m_i]$, and c is the number of classes. The dataset and labels across the entire network are denoted by $\mathbf{X} \triangleq (\mathbf{X}_1, \ldots, \mathbf{X}_N)$ and $\mathbf{Y} \triangleq (\mathbf{Y}_1, \ldots, \mathbf{Y}_N)$. The communication topology is decentralized (without a central server) as illustrated in Fig. 1. We consider a polynomial neural architecture [35]–[37], and use quadratic activations along with a mean-squared error loss. The goal is to learn a model $\mathbf{W}$ to minimize the empirical loss function:

$$\frac{1}{m}\sum_{i\in[m]} \mathcal{L}(\mathbf{W}; \mathbf{x}_i, \mathbf{y}_i) = \frac{1}{m}\sum_{i\in[m]} \|f(\mathbf{x}_i, \mathbf{W}) - \mathbf{y}_i\|_2^2 \qquad (1)$$

where $m \triangleq \sum_{i\in[N]} m_i$, and $f(\mathbf{x}_i, \mathbf{W})$ is the output of a feed-forward neural network consisting of L hidden layers, and a final classification layer (layer $L+1$) as shown in Fig. 2, at a single data point $\mathbf{x}_i$ with label $\mathbf{y}_i$ (i^{th} column of $\mathbf{X}$ and $\mathbf{Y}$). The model parameters connecting layer $l-1$ to layer l are given by a matrix $\mathbf{W}^l$ of size $d_l \times d_{l-1}$, where d_l is the number of neurons at layer $l \in [L+1]$, and $\mathbf{W} = (\mathbf{W}^1, \ldots, \mathbf{W}^{L+1})$. Accordingly, $d_0 = d$, and $d_{L+1} = c$. Training is done via gradient descent, where the model is updated iteratively as,

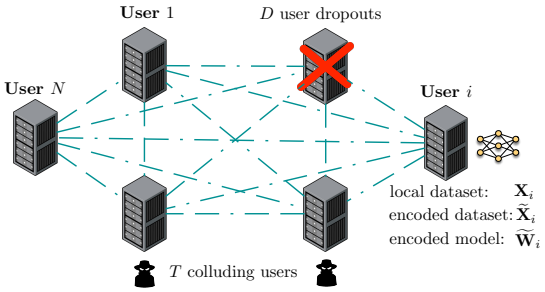

Fig. 1. **Collaborative learning model.** The multi-party learning setup of CLOVER. User $i \in [N]$ holds a local dataset $\mathcal{X}_i$, along with an encoded dataset $\widetilde{\mathbf{X}}_i$ and encoded model $\widetilde{\mathbf{W}} = (\widetilde{\mathbf{W}}^1, \ldots, \widetilde{\mathbf{W}}^{L+1})$.

$$\mathbf{W}(t+1) = \mathbf{W}(t) - \frac{\eta}{m}\sum_{i\in[m]} \nabla\mathcal{L}(\mathbf{W}(t); \mathbf{x}_i, \mathbf{y}_i) \qquad (2)$$

where $\nabla\mathcal{L}(\mathbf{W}(t); \mathbf{x}_i, \mathbf{y}_i)$ is the gradient for a single data point, $\mathbf{W}(t)$ is the estimated model parameters from training round t. Up to D users may drop out at each training round (e.g., due to poor channel conditions or device unavailability).

Threat Model. We consider an *honest-but curious* adversary model [1], [6], [7], [11], in which adversaries follow the protocol but try to obtain further information about the local datasets of honest users. Up to T are adversarial, who may collude with each other. The set of adversarial and honest users are denoted by $\mathcal{T}$ and $\mathcal{H}$.

Information-theoretic Privacy. Our goal is end-to-end information-theoretic privacy, where adversaries learn no information about the local datasets of honest users, beyond the final model [1], [10], [11]. Formally, this can be stated as,

$$I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{[N]\setminus\mathcal{T}}; \mathcal{M}_\mathcal{T} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \mathbf{W}(J)) = 0 \qquad (3)$$

for any $\mathcal{T}$ such that $|\mathcal{T}| \leq T$, where J is the total number of training rounds, and $\mathcal{M}_\mathcal{T}$ denotes the collection of all messages received or generated by the adversaries. Similar to [11], our framework is bound to finite field operations, where the datasets are represented in a finite field $\mathbb{F}_p$ of integers modulo a large prime p.

Main Problem. The state-of-the-art for PPML under (3) is the COPML framework [11], which leverages LCC [10] to partition the dataset $\mathbf{X}$ into K equal-sized shards, and encodes them using a Lagrange interpolation polynomial along with T random matrices. Each user then obtains an encoded dataset $\widetilde{\mathbf{X}}_i$, whose size is only $(1/K)^{th}$ of $\mathbf{X}$. Training is performed on the coded datasets, from which the final model $\mathbf{W}(J)$ can be recovered as long as $N \geq D + (\deg f)(K+T-1)+1$, where $\deg f$ quantifies the polynomial degree after J training rounds. K quantifies the *degree of parallelization*; each user needs to process only $(1/K)^{th}$ of the dataset $\mathbf{X}$, hence as the network size N grows, one can select a larger K for faster training. On the other hand, $\deg f$ grows exponentially after each multiplication operation (associated with gradient computations). To prevent a degree explosion, users have to perform an expensive *degree reduction* operation with a quadratic communication overhead after each training round. As a result, COPML can only be applied to simpler logistic/linear regression models, as opposed to more complex neural network training. Our goal is to address this challenge, where we ask the question,

- *How can we train a neural network to solve* (1) *with linear communication complexity, under multi-round information-theoretic privacy guarantees from* (3)*?*

Double Lagrange Coding. To address this challenge, we propose CLOVER, a privacy-preserving neural network training framework with linear communication complexity. Our key contribution is a novel degree reduction mechanism with linear communication complexity, as opposed to the quadratic state-of-the-art. At the outset, this mechanism generates two Lagrange polynomials, a higher degree polynomial used to decode a masked version of the true computations (while keeping their true values hidden), and a lower degree polynomial to re-encode them. In doing so, we decouple communication into online and offline phases, and offload the communication-intensive operations to the latter. We then propose an efficient offline randomness generation mechanism where the communication volume is inversely proportional to the number of users, through which we reduce the overall communication overhead (both online and offline) from quadratic to linear. We next describe the individual steps of CLOVER.

III. The CLOVER Framework

CLOVER consists of four key components described below.

1. Dataset Encoding. Users first agree on $N+K+T$ distinct public parameters $\{\alpha_j\}_{j\in[N]}$, $\{\beta_j\}_{j\in[K+T]}$ from $\mathbb{F}_p$. User $i\in[N]$ then partitions its local dataset $\mathbf{X}_i=(\mathbf{X}_{i1},\ldots,\mathbf{X}_{iK})$ into K submatrices of size $d\times\frac{m_i}{K}$, and generates a Lagrange polynomial of degree $K+T-1$:

$$u_i(z)\triangleq\sum_{k\in[K]}\mathbf{X}_{ik}\prod_{l\in[K+T]\setminus\{k\}}\frac{z-\beta_l}{\beta_k-\beta_l}+\sum_{k=K+1}^{K+T}\mathbf{V}_{ik}\prod_{l\in[K+T]\setminus\{k\}}\frac{z-\beta_l}{\beta_k-\beta_l}\tag{4}$$

where $u_i(\beta_k)=\mathbf{X}_{ik}$ for $k\in[K]$, and $\{\mathbf{V}_{ik}\}_{k=K+1}^{K+T}$ are uniformly random matrices from $\mathbb{F}_p$. Then, user i sends $\widetilde{\mathbf{X}}_{ij}\triangleq u_i(\alpha_j)$ to user $j\in[N]$. Finally, user i obtains the encoded dataset by concatenating the received $\{\widetilde{\mathbf{X}}_{ji}\}_{j\in[N]}$:

$$\widetilde{\mathbf{X}}_i\triangleq(\widetilde{\mathbf{X}}_{1i},\ldots,\widetilde{\mathbf{X}}_{Ni})\in\mathbb{F}_p^{d\times\frac{m}{K}}\tag{5}$$

The goal of dataset encoding is two-fold: 1) hide its content against adversaries, 2) reduce the size of data processed during training; each user computes the gradient on an encoded dataset $\widetilde{\mathbf{X}}_i$, whose size is $(1/K)^{th}$ of the original dataset $\mathbf{X}$. As the network size N increases, one can select a larger K, reducing the computation load per-user, speeding up training.

2. Label Encoding. Similarly, user $i\in[N]$ encodes the labels by partitioning $\mathbf{Y}_i=(\mathbf{Y}_{i1},\ldots,\mathbf{Y}_{iK})$ into K submatrices of size $c\times\frac{m_i}{K}$, and generating a Lagrange polynomial:

$$v_i(z)\triangleq\sum_{k\in[K]}\mathbf{Y}_{ik}\prod_{l\in[K+T]\setminus\{k\}}\frac{z-\beta_l}{\beta_k-\beta_l}+\sum_{k=K+1}^{K+T}\mathbf{U}_{ik}\prod_{l\in[K+T]\setminus\{k\}}\frac{z-\beta_l}{\beta_k-\beta_l}\tag{6}$$

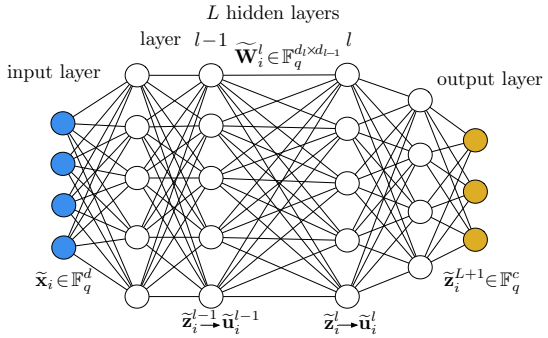

Fig. 2. **Neural architecture.** The neural network consists of an input layer, L hidden layers, and an output (classification) layer. The encoded model parameters connecting layer $l-1$ to layer l is given by $\widetilde{\mathbf{W}}_i^l\in\mathbb{F}_p^{d_l\times d_{l-1}}$ for user $i\in[N]$, where d_l is the number of neurons at layer l.

where $\{\mathbf{U}_{ik}\}_{k=K+1}^{K+T}$ are uniformly random matrices, and sends $\widetilde{\mathbf{Y}}_{ij}\triangleq v_i(\alpha_j)$ to user $j\in[N]$. Finally, user i obtains the encoded labels by concatenating the received $\{\widetilde{\mathbf{Y}}_{ji}\}_{j\in[N]}$:

$$\widetilde{\mathbf{Y}}_i\triangleq(\widetilde{\mathbf{Y}}_{1i},\ldots,\widetilde{\mathbf{Y}}_{Ni})\in\mathbb{F}_p^{c\times\frac{m}{K}}\tag{7}$$

3. Model Initialization. To preserve the privacy of intermediate computations, the model $\mathbf{W}(t)=(\mathbf{W}^1(t),\ldots,\mathbf{W}^{L+1}(t))$ at time $t=0$ should be initialized without revealing its true value to any user. To do so, users first agree on $N-T$ distinct public parameters $\lambda_1,\ldots,\lambda_{N-T}$ from $\mathbb{F}_p$. Then, user $i\in[N]$ generates $T+1$ uniformly random matrices $\mathbf{W}_i^l(0),\{\mathbf{U}_{ik}^l\}_{k\in\{K+1,\ldots,K+T\}}$ of size $\frac{d_l}{N-T}\times d_{l-1}$, forms a degree $K+T-1$ Lagrange polynomial,

$$q_i(z)\triangleq\mathbf{W}_i^l(0)\sum_{k\in[K]}\prod_{l\in[K+T]\setminus\{k\}}\frac{z-\beta_l}{\beta_k-\beta_l}+\sum_{k\in\{K+1,\ldots,K+T\}}\mathbf{U}_{ik}^l\prod_{l\in[K+T]\setminus\{k\}}\frac{z-\beta_l}{\beta_k-\beta_l}\tag{8}$$

and sends an encoded vector $\widetilde{\mathbf{W}}_{ij}^l(0)=q_i(\alpha_j)$ to user $j\in[N]$. After receiving $\widetilde{\mathbf{W}}_{1i}^l(0),\ldots,\widetilde{\mathbf{W}}_{Ni}^l(0)$, user $i\in[N]$ generates a (larger dimensional) coded matrix of size $d_l\times d_{l-1}$,

$$\widetilde{\mathbf{W}}_i^l(0)=\Big(\sum_{j\in[N]}\lambda_1^{j-1}\widetilde{\mathbf{W}}_{ji}^l(0)^{\mathrm{T}},\cdots,\sum_{j\in[N]}\lambda_{N-T}^{j-1}\widetilde{\mathbf{W}}_{ji}^l(0)^{\mathrm{T}}\Big)^{\mathrm{T}}$$

where the (randomly) initialized model is given by,

$$\mathbf{W}^l(0)\triangleq\Big(\sum_{j\in[N]}\lambda_1^{j-1}\mathbf{W}_j^l(0)^{\mathrm{T}},\cdots,\sum_{j\in[N]}\lambda_{N-T}^{j-1}\mathbf{W}_j^l(0)^{\mathrm{T}}\Big)^{\mathrm{T}}$$

whose true value is hidden by the T random matrices,

$$\mathbf{V}_k^l\triangleq\Big(\sum_{j\in[N]}\lambda_1^{j-1}\mathbf{U}_{jk}^{\mathrm{T}},\cdots,\sum_{j\in[N]}\lambda_{N-T}^{j-1}\mathbf{U}_{jk}^{\mathrm{T}}\Big)^{\mathrm{T}}\text{ for }k\in[T].$$

The key intuition is that, to generate a coded matrix of size $d_l\times d_{l-1}$, each user only sends a matrix of size $\frac{d_l}{N-T}\times d_{l-1}$ (which required a matrix of $d_l\times d_{l-1}$ in [11]). The final coded matrix is then generated by combining the *lower-dimensional* coded matrices. This stage is independent from data, thus can be carried out fully offline.

4. Gradient Computing and Model Update. Using the encoded datasets, users then compute the gradient and update the model. Note that from (4), each *coded* data point (a single

column of $\widetilde{\mathbf{X}}_i$) encodes K true data points (K columns of $\mathbf{X}$). We next present the gradient computation for a single coded data point denoted by $\widetilde{\mathbf{x}}_{bi}$, and label $\widetilde{\mathbf{y}}_{bi}$, which correspond to column $b \in [\frac{m}{K}]$ of $\widetilde{\mathbf{X}}_i$ and $\widetilde{\mathbf{Y}}_i$ respectively, at a single training round. For simplicity, we omit the round index t.

Gradient computation. Let $\mathbf{z}^l$ and $\mathbf{u}^l$ denote the input and output of the activation function at layer l, respectively, as illustrated in Fig. 2. Then, $\mathbf{z}^l = \mathbf{W}^l \mathbf{u}^{l-1}$ for $l \in [L+1]$, and $\mathbf{u}^l = g(\mathbf{z}^l)$ for $l \in [L]$, where $g(\cdot)$ is a quadratic activation function applied element-wise. Then, gradient computations consist of the following forward/backward propagation steps.

(Forward Propagation): User $i \in [N]$ initially computes,

$$\widetilde{\mathbf{z}}_i^l \triangleq \widetilde{\mathbf{W}}_i^l \widetilde{\mathbf{u}}_i^{l-1} \in \mathbb{F}_p^{d_l \times 1} \text{ for each layer } l \in [L+1], \quad (9)$$

where $\widetilde{\mathbf{u}}_i^0 \triangleq \widetilde{\mathbf{x}}_{bi}$. Note that $\widetilde{\mathbf{x}}_{bi}$ corresponds to (evaluations of) a degree $K+T-1$ polynomial, on the other hand, the degree of the resulting polynomial in (9) grows exponentially as the number of layers increase, which requires users to reduce the polynomial degree without breaching privacy. To address this, we introduce a novel degree reduction mechanism $\phi(\cdot)$,

$$(\widetilde{\mathbf{z}}_1^l, \ldots, \widetilde{\mathbf{z}}_N^l) \leftarrow \phi(\widetilde{\mathbf{z}}_1^l, \ldots, \widetilde{\mathbf{z}}_N^l) \quad (10)$$

as will be detailed in Section IV. At the end of (10), each user $i \in [N]$ receives an updated coded vector $\widetilde{\mathbf{z}}_i^l$ corresponding to a Lagrange polynomial with degree $K+T-1$, and computes,

$$\widetilde{\mathbf{u}}_i^{(l)} = g(\widetilde{\mathbf{z}}_i^l). \quad (11)$$

(Backward Propagation): Gradients are computed via backpropagation using the encoded dataset and labels. The computations are the same as conventional backpropagation [36], where user $i \in [N]$ locally computes,

$$\widetilde{\boldsymbol{\delta}}_i^l = \begin{cases} 2(\widetilde{\mathbf{z}}_i^{L+1} - \widetilde{\mathbf{y}}_{bi}) & \text{if } l = L+1 \\ 2diag(\widetilde{\boldsymbol{z}}_i^l) \times (\widetilde{\mathbf{W}}_i^{l+1})^T \times \widetilde{\boldsymbol{\delta}}_i^{l+1} & \text{otherwise} \end{cases} \quad (12)$$

such that $diag(\cdot)$ is a diagonal matrix where the j^{th} diagonal is equal to the j^{th} element of $\widetilde{\boldsymbol{z}}_i^l$. The key challenge is that, except for $l = L+1$, (12) corresponds to a high degree (degree $3(K+T-1)$) polynomial. To reduce the degree back to $K+T-1$, users need to carry out a degree reduction operation,

$$(\widetilde{\boldsymbol{\delta}}_1^l, \ldots, \widetilde{\boldsymbol{\delta}}_N^l) \leftarrow \phi(\widetilde{\boldsymbol{\delta}}_1^l, \ldots, \widetilde{\boldsymbol{\delta}}_N^l) \quad (13)$$

after evaluating (12) for each layer $l \in [L]$. At the end of (13), each user $i \in [N]$ receives a new coded vector $\widetilde{\boldsymbol{\delta}}_i^l$ corresponding to a Lagrange polynomial of degree $K+T-1$. Then, each user i can compute the gradient of $\mathbf{W}^l$ as,

$$\widetilde{\mathbf{G}}_{bi}^l \triangleq \widetilde{\boldsymbol{\delta}}_i^l (\widetilde{\mathbf{u}}^{l-1})^T \in \mathbb{F}_p^{d_l \times 1} \qquad \forall l \in [L+1]. \quad (14)$$

Model Update. After computing the gradients, users update the model according to (2). Note that (14) can be viewed as an evaluation of a degree $3(K+T-1)$ polynomial $r_b^l(\alpha)$ such that $r_b^l(\alpha_i) = \widetilde{\mathbf{G}}_{bi}^l$, whereas $r_b^l(\beta_k)$ for $k \in [K]$ refer to the true gradients from K data points. On the other hand, (2) requires the sum gradient $\sum_{i\in[m]} \nabla \mathcal{L}(\mathbf{W}(t); \mathbf{x}_i, \mathbf{y}_i) = \sum_{b\in[m/K], k\in[K]} r_b^l(\beta_k)$. In principle, one can collect the evaluations from at any set of $3(K+T-1)+1$ users to interpolate $r_b^l(\alpha)$ and decode the true gradients $r_b^l(\beta_k)$, then sum them up. However, revealing the gradients can breach the privacy of local datasets [38]. To address this, we propose a privacy-preserving model update mechanism with linear communication complexity (significantly improving the quadratic state-of-the-art [11]). The key intuition is to decode a *masked version* of the K true gradients for each *coded gradient* (where the true gradient is hidden by additive random masks), aggregate them, and re-encode them with a Lagrange polynomial, while simultaneously ensuring the cancellation of all additive masks. At the end, each user $i \in [N]$ obtains a coded gradient $\widetilde{\mathbf{G}}_i^l$ that encodes the *sum of all gradients* $\sum_{i\in[m]} \nabla \mathcal{L}(\mathbf{W}(t); \mathbf{x}_i, \mathbf{y}_i)$ from the individual data points with a Lagrange polynomial of degree $K+T-1$, using which each user updates the model as $\widetilde{\mathbf{W}}_i^l \leftarrow \widetilde{\mathbf{W}}_i^l - \frac{\eta}{m} \widetilde{\mathbf{G}}_i^l$ for all $l \in [L+1]$.

Finally, note that $\eta \ll 1$ in (2), whereas CLOVER is bound to finite field polynomial operations. To handle this, one can either consider a sufficiently large field size and treat all computations in the integer domain, or leverage the secure quantization protocol from [39] to reduce the required field size [11]. In our theoretical analysis, we consider the former, whereas we utilize the latter in our experiments.

Final Model Recovery. After J rounds, parties can collect the coded model $\widetilde{\mathbf{W}}_i(J)$ from any set of $K+T$ users, and recover the final model $\mathbf{W}(J)$ using polynomial interpolation.

IV. Double Lagrange Coding

We now introduce the details of our new degree reduction primitive for LCC. At the outset, this operation generates two Lagrange polynomials; a higher degree polynomial to decode a masked version of the true gradient computations, and a lower degree Lagrange polynomial to re-encode them. In doing so, we leverage an efficient shared randomness generation mechanism using MDS codes, which incurs only a linear communication complexity. Accordingly, we will call this mechanism *Double Lagrange Coding*.

Consider a polynomial $f(\cdot)$ of degree $\deg(f) = M$ for some $M > K+T-1$, where $f(\beta_1), \ldots, f(\beta_K) \in \mathbb{F}_p^{d_1 \times d_2}$ represent the K desired computations (e.g., gradient computations for K data points) for some d_1, d_2, and $f(\alpha_i)$ is the computation performed by user $i \in [N]$. The protocol then generates a new (low-degree) Lagrange polynomial $f'(\cdot)$ of degree $K+T-1$, such that $f'(\beta_k) = f(\beta_k)$ for all $k \in [K]$, and $f'(\beta_k) \in \mathbb{F}_p^{d_1 \times d_2}$ are uniformly random vectors for all $k \in \{K+1, \ldots, K+T\}$. At the end, each user $i \in [N]$ only learns an evaluation point $f'(\alpha_i)$, without learning any information about the true computations $\{f'(\beta_k)\}_{k\in[K]}$. As such, the new (low degree) polynomial preserves the K desired computation results from the old (higher degree) polynomial, without revealing any information about their true values.

Our degree reduction mechanism consists of offline (data-agnostic) and online (data-dependent) phases. The offline phase is independent from the data, hence can be carried out in advance when the network load is low. The online phase depends on data, and should be carried out after training starts.

(Offline) In the offline phase, users agree on $M+1$ distinct public parameters $\theta_1, \ldots, \theta_{M+1} \in \mathbb{F}_p$ such that $\theta_k = \beta_k$ for all $k \in [K]$. Each user $i \in [N]$ then generates $M+1$ uniformly

random matrices $\mathbf{R}_{i1}, \ldots, \mathbf{R}_{i,M+1}$ of size $\frac{d_1}{N-T} \times d_2$ from $\mathbb{F}_p$, and forms a Lagrange polynomial of degree M,

$$\psi_i(z) = \sum_{k\in[M+1]} \mathbf{R}_{ik} \prod_{l\in[M+1]\setminus\{k\}} \frac{z-\theta_l}{\theta_k-\theta_l} \tag{15}$$

where $\psi_i(\theta_k) = \mathbf{R}_{ik}$ for all $k \in [M+1]$. Then, user i sends an encoded vector,

$$\widetilde{\mathbf{R}}_{ij} \triangleq \psi_i(\alpha_j) \tag{16}$$

to user $j \in [N]$. In addition to (15), user i also creates a second (lower degree) Lagrange polynomial with degree $K+T-1$,

$$\sigma_i(z) = \sum_{k\in[K]} \mathbf{R}_{ik} \prod_{l\in[K+T]\setminus\{k\}} \frac{z-\beta_l}{\beta_k-\beta_l} + \sum_{k\in\{K+1,\ldots,K+T\}} \mathbf{Q}_{ik} \prod_{l\in[K+T]\setminus\{k\}} \frac{z-\beta_l}{\beta_k-\beta_l} \tag{17}$$

where $\mathbf{Q}_{ik} \in \mathbb{F}_p^{\frac{d_1}{N-T}\times d_2}$ are generated uniformly random for $k \in \{K+1, \ldots, K+T\}$. Then, user i sends an encoded vector,

$$\overline{\mathbf{R}}_{ij} \triangleq \sigma_i(\alpha_j) \tag{18}$$

to user $j \in [N]$. After receiving $\{\widetilde{\mathbf{R}}_{ji}, \overline{\mathbf{R}}_{ji}\}_{j\in[N]}$, user $i \in [N]$ combines them to generate two new Lagrange polynomials,

$$\widetilde{\mathbf{R}}_i \triangleq \Big(\sum_{j\in[N]} \lambda_1^{j-1} \widetilde{\mathbf{R}}_{ji}^{\mathrm{T}}, \cdots, \sum_{j\in[N]} \lambda_{N-T}^{j-1} \widetilde{\mathbf{R}}_{ji}^{\mathrm{T}} \Big)^{\mathrm{T}} \tag{19}$$

$$= \sum_{k\in[M+1]} \mathbf{R}_k \prod_{l\in[M+1]\setminus\{k\}} \frac{\alpha_i-\theta_l}{\theta_k-\theta_l} \tag{20}$$

$$\overline{\mathbf{R}}_i \triangleq \Big(\sum_{j\in[N]} \lambda_1^{j-1} \overline{\mathbf{R}}_{ji}^{\mathrm{T}}, \cdots, \sum_{j\in[N]} \lambda_{N-T}^{j-1} \overline{\mathbf{R}}_{ji}^{\mathrm{T}} \Big)^{\mathrm{T}} \tag{21}$$

$$= \sum_{k\in[K]} \mathbf{R}_k \prod_{l\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_l}{\beta_k-\beta_l} + \sum_{k\in\{K+1,\ldots,K+T\}} \mathbf{Q}_k \prod_{l\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_l}{\beta_k-\beta_l} \tag{22}$$

where $\mathbf{R}_k \triangleq (\sum_{j\in[N]} \lambda_1^{j-1} \mathbf{R}_{jk}^{\mathrm{T}}, \cdots, \sum_{j\in[N]} \lambda_{N-T}^{j-1} \mathbf{R}_{jk}^{\mathrm{T}})^{\mathrm{T}}$, and $\mathbf{Q}_k \triangleq (\sum_{j\in[N]} \lambda_1^{j-1} \mathbf{Q}_{jk}^{\mathrm{T}}, \cdots, \sum_{j\in[N]} \lambda_{N-T}^{j-1} \mathbf{Q}_{jk}^{\mathrm{T}})^{\mathrm{T}}$. In doing so, the key motivation is to generate high dimensional shared randomness using a lower dimensional random polynomial generated by each user. Specifically, the dimension of the random vectors generated (and embedded in a Lagrange polynomial) by each user has size $\frac{d_1}{N-T} \times d_2$, whereas the size of the random vectors embedded in the Lagrange polynomials from (19) and (21) both have size $d_1 \times d_2$.

(Online) In the online phase, each user $i \in [N]$ broadcasts $f(\alpha_i) - \widetilde{\mathbf{R}}_i$, which can be viewed as an evaluation of a degree M monomial $h(z) = f(z) - \psi(z)$ at point $z = \alpha_i$, where

$$h(\alpha_i) = f(\alpha_i) - \psi(\alpha_i) = f(\alpha_i) - \widetilde{\mathbf{R}}_i \qquad \forall i \in [N], \tag{23}$$

$$h(\beta_k) = f(\beta_k) - \psi(\beta_k) = f(\beta_k) - \mathbf{R}_k \qquad \forall k \in [K] \tag{24}$$

such that the desired computations $f(\beta_k)$ are hidden by an additive random mask $\mathbf{R}_k = \psi(\beta_k)$, where

$$\psi(z) \triangleq \sum_{k\in[M+1]} \mathbf{R}_k \prod_{l\in[M+1]\setminus\{k\}} \frac{z-\theta_l}{\theta_k-\theta_l} \tag{25}$$

Then, after receiving $h(\alpha_i)$ from a set $\mathcal{I}$ of at least $|\mathcal{I}| \geq M+1$ users, every user can interpolate the polynomial $h(z)$, and compute $h(\beta_k) = f(\beta_k) - \psi(\beta_k)$ for all $k \in [K]$. Finally, each user $i \in [N]$ re-encodes the desired computations $\{f(\beta_k)\}_{k\in[K]}$ with a degree $K+T-1$ Lagrange polynomial,

$$f'(\alpha_i) = \sum_{k\in[K]} h(\beta_k) \prod_{k\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_l}{\beta_k-\beta_l} + \overline{\mathbf{R}}_i \tag{26}$$

$$= \sum_{k\in[K]} f(\beta_k) \prod_{k\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_l}{\beta_k-\beta_l} + \sum_{k\in\{K+1,\ldots,K+T\}} \mathbf{V}_k \prod_{l\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_l}{\beta_k-\beta_l} \tag{27}$$

V. Theoretical Analysis

We now present the formal privacy and complexity guarantees of CLOVER.

Theorem 1. *(Information-theoretic privacy) CLOVER guarantees information theoretic privacy:*

$$I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_{\mathcal{T}} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \mathbf{W}(J)) = 0 \tag{28}$$

against any set $\mathcal{T}$ of $|\mathcal{T}| \leq T$ adversaries, where $\mathcal{M}_{\mathcal{T}}$ is the collection of all messages received/generated by adversaries.

In the sequel, we let $m_i = \bar{m}$ for all $i \in [N]$, to present the complexity explicitly with respect to the number of users.

Theorem 2. *(Communication complexity) The per-user communication complexity of CLOVER is $O(\frac{(d+c)N\bar{m}}{K} + \frac{JN\bar{m}}{K} \sum_{l\in[L+1]} d_l d_{l-1})$ in the online phase, and $O(\frac{JN^2\bar{m}}{(N-T)K} \sum_{l\in[L+1]} d_l d_{l-1})$ in the offline phase, respectively. With $T = O(N)$ and $K = \Theta(N)$, the total communication complexity across all N users (including both online and offline phases) is linear in the number of users, which is $O(N(d+c)\bar{m} + NJ\bar{m} \sum_{l\in[L+1]} d_l d_{l-1})$.*

The *recovery threshold* is defined as the minimum number of users required to correctly decode the final model.

Theorem 3. *(Recovery threshold) The recovery threshold of CLOVER is $N \geq D + 3(K+T-1) + 1$.*

Proof. The recovery threshold is determined by the minimum number of local computations required for polynomial interpolation, which is $N - D \geq 3(K+T-1)+1$ from Section III. □

The recovery threshold of COPML is $N \geq D + (2r+1)(K+T-1)+1$, $r \geq 1$ [11], hence CLOVER provides equal adversary tolerance T, dropout resilience D, and parallelization gain K, while also cutting the communication cost.

VI. Conclusion

This work presents CLOVER, the first collaborative PPML framework with linear communication complexity, under the information-theoretic privacy setting. CLOVER significantly reduces the communication overhead of the state-of-the-art, while achieving equal adversary and dropout resilience.

REFERENCES

[1] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *38th IEEE Symposium on Security and Privacy*. IEEE, 2017, pp. 19–38.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC'88)*, 1988, p. 1–10.

[3] Z. Beerliová-Trubíniová and M. Hirt, "Perfectly-secure MPC with linear communication complexity," in *Theory of Cryptography Conference*. Springer, 2008, pp. 213–230.

[4] M. Al-Rubaie and J. M. Chang, "Privacy-preserving machine learning: Threats and solutions," *IEEE Security & Privacy*, vol. 17, no. 2, pp. 49–58, 2019.

[5] I. Damgård and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," in *Annual International Cryptology Conference*. Springer, 2007, pp. 572–590.

[6] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *IEEE Symposium on Security and Privacy*, 2013, pp. 334–348.

[7] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans, "Privacy-preserving distributed linear regression on high-dimensional data," *Proceedings on Privacy Enhancing Tech.*, vol. 2017, no. 4, pp. 345–364, 2017.

[8] P. Mohassel and P. Rindal, "ABY 3: A mixed protocol framework for machine learning," in *ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 35–52.

[9] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: Efficient and private neural network training," Cryptology ePrint Archive, Report 2018/442, 2018.

[10] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *The 22nd International Conference on Artificial Intelligence and Statistics (AISTATS 2019)*. PMLR, 2019, pp. 1215–1225.

[11] J. So, B. Güler, and S. Avestimehr, "A scalable approach for privacy-preserving collaborative machine learning," in *Advances in Neural Information Processing Systems: Annual Conference on Neural Information Processing Systems, NeurIPS*, Dec. 2020.

[12] J. So, B. Güler, and A. S. Avestimehr, "Codedprivateml: A fast and privacy-preserving framework for distributed machine learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 1, pp. 441–451, 2021.

[13] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography Conference*. Springer, 2006, pp. 265–284.

[14] K. Chaudhuri and C. Monteleoni, "Privacy-preserving logistic regression," in *Adv. in Neural Inf. Proc. Sys.*, 2009, pp. 289–296.

[15] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1310–1321.

[16] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 308–318.

[17] M. Pathak, S. Rane, and B. Raj, "Multiparty differential privacy via aggregation of locally trained classifiers," in *Advances in Neural Inf. Processing Systems*, 2010, pp. 1876–1884.

[18] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," in *Int. Conf. on Learning Representations*, 2018.

[19] A. Rajkumar and S. Agarwal, "A differentially private stochastic gradient descent algorithm for multiparty classification," in *Int. Conf. on Artificial Intelligence and Statistics (AISTATS'12)*, vol. 22, La Palma, Canary Islands, Apr 2012, pp. 933–941.

[20] B. Jayaraman, L. Wang, D. Evans, and Q. Gu, "Distributed learning without distress: Privacy-preserving empirical risk minimization," *Advances in in Neural Information Processing Systems*, pp. 6346–6357, 2018.

[21] W.-N. Chen, A. Ozgur, and P. Kairouz, "The poisson binomial mechanism for unbiased federated learning with secure aggregation," in *International Conference on Machine Learning*. PMLR, 2022, pp. 3490–3506.

[22] W.-N. Chen, C. A. C. Choo, P. Kairouz, and A. T. Suresh, "The fundamental price of secure aggregation in differentially private federated learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 3056–3089.

[23] P. Kairouz, Z. Liu, and T. Steinke, "The distributed discrete gaussian mechanism for federated learning with secure aggregation," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5201–5212.

[24] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*. Stanford University, Stanford, 2009, vol. 20, no. 09.

[25] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[26] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Int. Conf. on Machine Learning*, 2016, pp. 201–210.

[27] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," *arXiv:1711.05189*, 2017.

[28] T. Graepel, K. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," in *Int. Conf. on Information Security and Cryptology*. Springer, 2012, pp. 1–21.

[29] J. Yuan and S. Yu, "Privacy preserving back-propagation neural network learning made practical with cloud computing," *IEEE Trans. on Parallel and Dist. Sys.*, vol. 25, no. 1, pp. 212–221, 2014.

[30] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network." *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 35, 2017.

[31] P. Li, J. Li, Z. Huang, C.-Z. Gao, W.-B. Chen, and K. Chen, "Privacy-preserving outsourced classification in cloud computing," *Cluster Computing*, pp. 1–10, 2017.

[32] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon, "Logistic regression model training based on the approximate homomorphic encryption," *BMC medical genomics*, vol. 11, no. 4, p. 83, 2018.

[33] 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*, vol. 30, no. 12, pp. 2381–2393, Dec 2018.

[34] K. Han, S. Hong, J. H. Cheon, and D. Park, "Logistic regression on homomorphic encrypted data at scale," *Annual Conf. on Innovative App. of Artificial Intelligence (IAAI-19)*, 2019.

[35] J. Kileel, M. Trager, and J. Bruna, "On the expressive power of deep polynomial neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[36] J. Shao, Y. Sun, S. Li, and J. Zhang, "Dres-fl: Dropout-resilient secure federated learning for non-iid clients via secret data sharing," in *Advances in Neural Information Processing Systems: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2022.

[37] G. G. Chrysos, S. Moschoglou, G. Bouritsas, J. Deng, Y. Panagakis, and S. Zafeiriou, "Deep polynomial neural networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 8, pp. 4021–4034, 2021.

[38] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1322–1333.

[39] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *International Conference on Financial Cryptography and Data Security*. Springer, 2010, pp. 35–50.