Scalable Multi-Round Multi-Party Privacy-Preserving Neural Network Training

Xingyu Lu, Umit Yigit Basaran, and Başak Güler, *Member, IEEE*

***Abstract*— Privacy-preserving machine learning has achieved breakthrough advances in collaborative training of machine learning models, under strong information-theoretic privacy guarantees. Despite the recent advances, communication bottleneck still remains as a major challenge against scalability in neural networks. To address this challenge, this paper presents the first scalable 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. Our contribution is an iterative coded computing mechanism with linear communication complexity, termed *Double Lagrange Coding*, which allows iterative scalable multi-party polynomial computations without degrading the parallelization gain, adversary tolerance, and dropout resilience throughout the iterations. While providing strong multi-round information-theoretic privacy guarantees, our framework achieves equal adversary tolerance, resilience to user dropouts, and model accuracy to the state-of-the-art, while reducing the communication overhead from quadratic to linear. In doing so, our framework addresses a key technical challenge in collaborative privacy-preserving machine learning, while paving the way for large-scale privacy-preserving iterative algorithms for deep learning and beyond.**

***Index Terms*— Privacy-preserving machine learning, collaborative training, information-theoretic privacy.**

I. Introduction

PRIVACY-PRESERVING collaborative machine learning (PPML) is a popular paradigm for joint training of machine learning (ML) models across multiple data-owners (users), without compromising the privacy of local data [1], [2], [3], [4], [5], [6], [7], [8], [9]. Recently, information and coding theoretic mechanisms has led to promising advances in the design of PPML frameworks [10], [11], [12]. This approach, known as *privacy-preserving coded computing*, first *encodes* the datasets and the model using a Lagrange interpolation polynomial, through Lagrange Coded Computing (LCC) [10]. The training computations are then performed using the encoded datasets and model as opposed to the true datasets, but as if they were performed on the true datasets. In doing so, the encoding operation injects randomness and computational redundancy across the local computations performed by different users, to provide strong information-theoretic privacy guarantees and resilience to user dropouts. The additional randomness is *reversible*; after multiple training rounds, the final model can be correctly recovered using polynomial interpolation using the computations performed on the encoded data. Accordingly, coded computing can provide strong information-theoretic privacy guarantees for the sensitive user data as well as resilience against user drop-outs, while achieving an order-of-magnitude speed-up in the training time compared to state-of-the-art cryptographic baselines [11], [12].

The major challenge against the scalability of such 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* 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*, leading to 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, where users learn no information beyond the final model, are applied to simpler logistic and linear regression tasks, as opposed to more complex neural network training.

To address this challenge, in this work we introduce a privacy-preserving distributed neural network training mechanism CLOVER (**C**ol**l**ab**o**rative pri**v**ate n**e**ural network t**r**aining), the first scalable 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, termed Double Lagrange Coding (DLC), which reduces the quadratic

Manuscript received 29 July 2023; revised 12 April 2024; accepted 5 August 2024. Date of publication 9 August 2024; date of current version 22 October 2024. This work was supported in part by OUSD (R&E)/RT&L under Agreement W911NF-20-2-0267, in part by NSF CAREER under Award CCF-2144927, and in part by UCR OASIS Fellowship. An earlier version of this paper was presented at the 2023 IEEE International Symposium on Information Theory (ISIT'23) [DOI: 10.1109/ISIT54713.2023.10206617]. *(Corresponding author: Başak Güler.)*

The authors are with the Department of Electrical and Computer Engineering, University of California at Riverside, Riverside, CA 92521 USA (e-mail: xlu065@ucr.edu; ubasa001@ucr.edu; bguler@ece.ucr.edu).

Communicated by A. Sarwate, Associate Editor for Security and Privacy.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIT.2024.3441509.

Digital Object Identifier 10.1109/TIT.2024.3441509

communication overhead of degree reduction 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, such as randomness generation for coding, which can be carried out in advance when network load is low, or in parallel with other components of training. 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, which reduces the communication volume via a layered coding mechanism using MDS (Maximum Distance Separable) matrices.[1] The total number of communicated variables is inversely proportional to the number of users, resulting in 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 conventional approaches, which, as a result, are limited to simpler logistic and linear regression, as opposed to neural networks, while achieving the same adversary and dropout resilience. In our experiments, we also implement CLOVER over a distributed multi-user network for image classification, and demonstrate its numerical performance in terms of the communication overhead, model accuracy, and training time. Our contributions can be summarized as follows:

- We propose CLOVER, the first privacy-preserving multi-party neural network framework with linear communication complexity, under strong end-to-end information-theoretic privacy guarantees.
- We introduce the first scalable degree reduction mechanism for Lagrange Coded Computing (LCC) with linear communication complexity, allowing scalable iterative coded computations, which can open up further research in privacy-preserving applications of iterative algorithms, beyond machine learning.
- We present the formal information-theoretic privacy guarantees of CLOVER for end-to-end multi-round 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.

II. Related Work

Beyond LCC, notable coded computing mechanisms that can be applied to decentralized machine learning tasks include the generalized PolyDot codes [13], to enhance the resilience of neural network training in the presence of error-prone and unreliable nodes. Reference [14] extends the generalized PolyDot codes to secure matrix multiplication, by introducing secure generalized PolyDot codes for server-worker computation offloading. In this setting, a trusted server offloads encoded computations to multiply two sensitive matrices to a set of honest-but-curious workers. In doing so, a novel secure coded computing mechanism is proposed to allow flexible communication loads, while preserving the privacy of sensitive matrices. Different from the server-worker data offloading setting, our focus is on the collaborative learning setting without a trusted party, in which no party can observe the datasets of other parties, hence the security of the datasets and intermediate model parameters should also be preserved during encoding.

Beyond coded computing, there are three complementary approaches to PPML: 1) Secure Multi-party Computing (MPC), 2) Differential Privacy, 3) Homomorphic Encryption. Secure MPC protocols for PPML build on a cryptographic primitive known as *secret sharing* [15], where parties inject randomness to sensitive data before sharing it with others, and then the training computations are performed on the secret shared data [2], [3], [4]. The randomness is reversible; after multiple training rounds, parties can recover the true model as if it was computed on the original data, by using the computations performed on secret shared data. In doing so, secure MPC protocols can preserve model accuracy and strong information-theoretic privacy [6], [7], [8], [9]. On the other hand, two challenges hinder scalability to larger networks beyond $3-4$ users: 1) they do not benefit from parallelization and distributed implementation for the computation load of training, in particular, the computation load at each user is as high as centralized training, where the local datasets of all users are pooled at a single location, 2) they require extensive interaction and communication between the users.

As a result, current secure MPC frameworks are primarily used on a *per-round* basis during training, as opposed to end-to-end multi-round training. This is known as *secure aggregation* in federated and distributed learning, where parties perform training locally on their local datasets, and then the local updates (e.g., local gradients) are aggregated using a secure MPC protocol, coordinated by a central server. In doing so, parties learn the aggregate (sum) of local gradients/models after each (global) training round, without observing them in the clear [16], [17], [18], [19]. On the other hand, secure aggregation reveals the aggregated gradients and the updated model after each training round, and privacy degrades as the number of rounds increase [20]. Moreover, secure aggregation protocols are vulnerable to multi-round privacy attacks [21], [22]. In contrast, in this work our focus is on *end-to-end multi-round* PPML, in which users can only learn the final model at the end of training, after multiple (global) training rounds, and no intermediate model or gradient can be revealed, even in aggregated form, at any intermediate training round. In doing so, CLOVER reveals no intermediate model or gradient during training, preventing such multi-round privacy degradation throughout the training.

[1]Throughout the manuscript, an *MDS matrix* refers to the generator matrix of an MDS code.

TABLE I
LIST OF NOTATIONS

N	Total number of users.
$\mathcal{H}$	The set of honest users.
$\mathcal{T} = [N]\backslash\mathcal{H}$	The set of adversarial users.
T	Maximum number of adversaries.
D	Maximum number of user dropouts at any given training round.
d	Number of features for a given data sample.
c	Number of classes for each data sample.
m	Number of local data samples held by each user $i \in [N]$.
$\mathbf{X}_i$	$d \times m$ matrix denoting the local dataset (features) of user $i \in [N]$.
$\mathbf{Y}_i$	$c \times m$ matrix denoting the local labels of user $i \in [N]$.
$\mathbf{X} = [\mathbf{X}_1 \ \cdots \ \mathbf{X}_N]$	$d \times Nm$ matrix denoting the local datasets (features) from all N users.
$\mathbf{Y} = [\mathbf{Y}_1 \ \cdots \ \mathbf{Y}_N]$	$c \times Nm$ matrix denoting the collection of the local labels from all N users.
K	Parallelization degree.
L	Number of hidden layers in the neural network.
d_l	Number of neurons at layer $l \in [L+1]$, with $d_0 = d$ and $d_{L+1} = c$.
$\mathbf{W}_l(t)$	True model parameters for layer $l \in [L+1]$ at training round t.
$\mathbf{G}_l(t)$	True gradient parameters for layer $l \in [L+1]$ at training round t.
$\widetilde{\mathbf{X}}_i$	Encoded dataset (features) for user $i \in [N]$.
$\widetilde{\mathbf{Y}}_i$	Encoded labels for user $i \in [N]$.
$\widetilde{\mathbf{W}}_{l,i}(t)$	Encoded model at user $i \in [N]$ for layer $l \in [L+1]$ at round t.
$\widetilde{\mathbf{G}}_{l,i}(t)$	Encoded gradient at user $i \in [N]$ for layer $l \in [L+1]$ at round t.
$\mathbb{F}_p$	Finite field of integers modulo a large prime p.
η	Learning rate.
J	Total number of training rounds.

Differential Privacy (DP) is a noisy release mechanism which protects the privacy of personally identifiable information by injecting irreversible noise during training. Beyond secure MPC and information-theoretic PPML frameworks, where the final model is revealed to the users, DP can further prevent privacy leakage from the final model, such that an adversary who has access to the final model cannot backtrack an individual's sensitive data [23], [24], [25], [26], [27], [28], [29], [30]. In doing so, DP leads to an inherent accuracy-privacy trade-off; stronger privacy guarantees require a higher noise level. In distributed settings, the noise accumulates, degrading model privacy. As a result, recent DP mechanisms have been integrated with information-theoretic PPML frameworks, which is known as distributed DP, to reduce the amount of noise accumulated in distributed settings, and increase model accuracy [31], [32], [33]. Though beyond the scope of our current work, we note that our approach can in principle also be combined with and benefit DP, which is an interesting future direction.

Homomorphic Encryption (HE) protocols allow computations to be performed on encrypted data [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44]. In doing so, they can tolerate a larger number of adversaries compared to secure MPC. The corresponding trade-off is that adversaries have bounded computational power, as opposed to information-theoretic frameworks where adversaries can have unbounded computational power. In addition, privacy guarantees are based on computational hardness assumptions, and stronger privacy guarantees require larger encrypted data size, increasing the computation load per user. As a result, HE is primarily utilized for inference tasks in machine learning, as opposed to computationally-intensive training.

III. PROBLEM FORMULATION

A. Notation and Preliminaries

We first introduce the notation that will be used throughout the paper. In the following, $\mathbf{x}$ denotes a vector, and $\mathbf{X}$ stands for a matrix. $\mathcal{X}$ denotes a set with cardinality $|\mathcal{X}|$, whereas $[N]$ represents the set $\{1, \ldots, N\}$. $\mathbf{X}^{\mathrm{T}}$ denotes the matrix transpose, whereas $tr(\mathbf{X})$ denotes the trace of $\mathbf{X}$, and $\mathbf{X}[s]$ denotes the s^{th} column of $\mathbf{X}$. $\|\mathbf{X}\|_F$ denotes the Frobenius norm, and $\odot$ is the Hadamard product. $\mathbb{F}_p$ denotes the finite field of integers modulo a large prime p.

Finally, $[x]_i$ denotes the secret sharing of a secret x by using Shamir's T-out-of-N Secret Sharing (SSS) protocol [15]. SSS embeds a secret x in a degree T polynomial,

$$f(\alpha) = x + \alpha r_1 + \ldots + \alpha^T r_T \tag{1}$$

where each coefficient $\{r_k\}_{k\in[T]}$ is generated independently and uniformly at random from $\mathbb{F}_p$. Then, user $i \in [N]$ receives a secret share denoted as $[x]_i \triangleq f(\alpha_i)$. SSS provides information-theoretic privacy for the secret against any set of T colluding users. The secret x can be reconstructed from any collection of $T+1$ shares using polynomial interpolation, but no information can be revealed from any group of T or fewer shares. Table I provides the list of key notations used in the remainder of our paper. We next introduce our system model.

B. Multi-Party Neural Network Training

We consider a collaborative neural network training task in a network of N users. Our framework is bound to finite field operations, where all training operations are carried out in a finite field $\mathbb{F}_p$ of integers modulo a large prime p. Similar to [1] and [11], the datasets are represented in the finite field as described in App. A. User i holds a local dataset represented

by a matrix $\mathbf{X}_i \in \mathbb{F}_p^{d\times m}$, where the k^{th} column denotes the feature vector for a single data sample $k \in [m]$, m denotes the number of local data samples held by each user,[2] and d denotes the number of features for each sample. The corresponding labels are represented by a binary matrix $\mathbf{Y}_i \in \mathbb{F}_p^{c\times m}$ for user $i \in [N]$, where the k^{th} column is the binary one-hot label vector for data sample $k \in [m]$, and c is the number of classes. The dataset and labels across the entire network are represented by $\mathbf{X} \triangleq \begin{bmatrix}\mathbf{X}_1 & \cdots & \mathbf{X}_N\end{bmatrix} \in \mathbb{F}_p^{d\times Nm}$ and $\mathbf{Y} \triangleq \begin{bmatrix}\mathbf{Y}_1 & \cdots & \mathbf{Y}_N\end{bmatrix} \in \mathbb{F}_p^{c\times Nm}$.

For the neural network architecture, we consider a polynomial neural network as given in Fig. 1, with L hidden layers, along with a final classification layer denoted by layer $L+1$, and quadratic activation functions $g(x) = x^2$ along with mean squared error loss [45]. The model parameters connecting layer $l-1$ to layer l are denoted by a matrix $\mathbf{W}_l \in \mathbb{F}_p^{d_l\times d_{l-1}}$, where d_l is the number of neurons at layer $l \in [L+1]$, with $d_0 \triangleq d$, and $d_{L+1} \triangleq c$. The input and output of the activation function at layer $l \in [L+1]$ is denoted by $\mathbf{Z}_l \in \mathbb{F}_p^{d_l\times Nm}$ and $\mathbf{U}_l \in \mathbb{F}_p^{d_l\times Nm}$, respectively, where,

$$\mathbf{Z}_l \triangleq \mathbf{W}_l\mathbf{U}_{l-1}, \tag{2}$$

such that $\mathbf{U}_0 \triangleq \mathbf{X}$, and

$$\mathbf{U}_l \triangleq g(\mathbf{Z}_l) \quad \forall l \in [L], \tag{3}$$

where the activation function $g(\cdot)$ is applied element-wise in (3). The goal is to learn the model parameters that minimize the empirical loss function,

$$\mathbf{W}_1^*, \ldots, \mathbf{W}_{L+1}^* = \arg\min_{\mathbf{W}_1,\ldots,\mathbf{W}_{L+1}} \frac{1}{Nm}\mathcal{L}(\mathbf{W}_1, \ldots, \mathbf{W}_{L+1}; \mathbf{X}, \mathbf{Y}) \tag{4}$$

where

$$\mathcal{L}(\mathbf{W}_1, \ldots, \mathbf{W}_{L+1}; \mathbf{X}, \mathbf{Y}) \triangleq \|\mathbf{Z}_{L+1} - \mathbf{Y}\|_F^2,$$

such that $\mathbf{Z}_{L+1} \in \mathbb{F}_p^{c\times Nm}$ is the output of the neural network at layer $L+1$. Training is carried out iteratively via gradient descent. At each training round t, users compute a gradient,

$$\mathbf{G}_l(t) \triangleq \nabla\mathcal{L}(\mathbf{W}_1, \ldots, \mathbf{W}_{L+1}; \mathbf{X}, \mathbf{Y})$$

for layer $l \in [L+1]$ by using the current state of the model $\mathbf{W}_1(t), \ldots, \mathbf{W}_{L+1}(t)$. After computing the gradient, the model is updated for the next training round,

$$\mathbf{W}_l(t+1) = \mathbf{W}_l(t) - \frac{\eta}{Nm}\mathbf{G}_l(t) \quad \forall l \in [L+1] \tag{5}$$

where $\mathbf{W}_l(t)$ denotes the estimated model parameters for layer l at training round t, and $\mathbf{G}_l(t)$ denotes the aggregated gradient across all Nm data samples for layer l. We consider a decentralized communication topology without a central server as illustrated in Fig. 2. At each training round, up to D users may drop out due to various reasons such as poor wireless connectivity, low battery, or device unavailability.

[2]For ease of exposition, we consider an equal number of local data samples held by each user. Without loss of generality, our techniques can also be extended to the scenario when the number of samples held by each user is different.

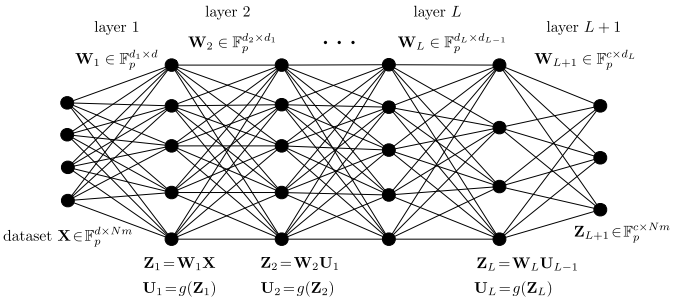


Fig. 1. Neural network model.

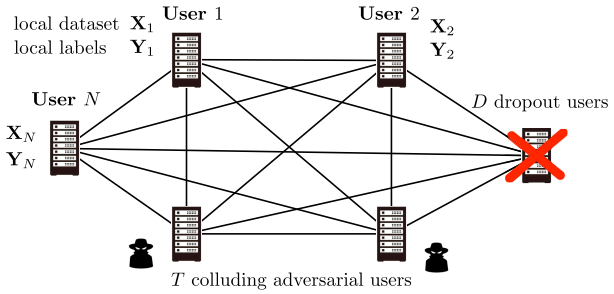


Fig. 2. Multi-party collaborative learning setup. User $i \in [N]$ holds a local dataset $\mathbf{X}_i$, along with the labels $\mathbf{Y}_i$.

Remark 1: Polynomial neural networks are primarily motivated by the Stone-Weierstrass polynomial approximation theorem [46]. In particular, for any continuous function $f(x)$ on a closed interval $x \in [a, b]$, for any $\epsilon > 0$, there exists a polynomial $\hat{f}(x)$ such that $|\hat{f}(x) - f(x)| \leq \epsilon$. The result can also be extended to multivariate polynomials [47].

C. Threat Model and Information-Theoretic Privacy

We consider an *honest-but curious* adversary model, where adversaries follow the protocol but try to obtain further information about the local datasets of honest users, using the messages exchanged throughout the protocol, which is the most common threat model in PPML [1], [6], [7], [11]. From N users, up to T users are adversarial, who may collude with each other. The set of adversarial and honest users are denoted by $\mathcal{T}$ and $\mathcal{H}$, respectively.

Our focus in this work is on 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]. This condition can formally be stated as follows,

$$I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{[N]\setminus\mathcal{T}}; \mathcal{M}_\mathcal{T} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) = 0 \tag{6}$$

for all $\mathcal{T}$ 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.

D. Main Problem

In this work, our goal is to develop scalable mechanisms to train the neural network model $\mathbf{W}_1, \ldots, \mathbf{W}_{L+1}$ from (4) under the information-theoretic privacy guarantees from (6).

The conventional approach for model training with end-to-end information-theoretic privacy is to leverage a combination of LCC and Shamir's Secret Sharing (SSS) [11]. LCC enables reducing the computational load for training, such that each user performs training on an encoded dataset whose size is only $(1/K)^{th}$ of the true dataset $\mathbf{X}$, where K quantifies the *degree of parallelization*. As the network size N grows, one can select a larger K for faster training. On the other hand, SSS enables secure encoding of the datasets and model for LCC, to prevent users from learning the true model and the datasets of other parties during encoding.

This approach can be applied to neural network training through the following steps. Initially, each user $i \in [N]$ secret shares its local dataset $\mathbf{X}_i \in \mathbb{F}_p^{d\times m}$ using SSS, by sending a share $[\mathbf{X}_i]_j \in \mathbb{F}_p^{d\times m}$ to user $j \in [N]$. User j concatenates and partitions the received shares into K equal-sized shards $\big[[\mathbf{X}_1]_j \ \cdots \ [\mathbf{X}_N]_j\big] = \big[[\overline{\mathbf{X}}_1]_j \ \cdots \ [\overline{\mathbf{X}}_K]_j\big]$, and sends to each user $i \in [N]$ an encoded matrix,

$$[\widetilde{\mathbf{X}}_i]_j = \sum_{k\in[K]} [\overline{\mathbf{X}}_k]_j \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=K+1}^{K+T} [\mathbf{V}_k]_j \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}}$$

where $\mathbf{V}_{K+1},\dots,\mathbf{V}_{K+T} \in \mathbb{F}_p^{d\times \frac{Nm}{K}}$ are T uniformly random matrices secret shared by a trusted crypto-service provider. Upon receiving $\{[\widetilde{\mathbf{X}}_i]_j\}_{j\in[N]}$, user i recovers the encoded dataset $\widetilde{\mathbf{X}}_i$ using polynomial interpolation. The encoded dataset $\widetilde{\mathbf{X}}_i \in \mathbb{F}_p^{d\times \frac{Nm}{K}}$ can be viewed as an interpolation point of a degree $K+T-1$ Lagrange polynomial,

$$x(\alpha) = \sum_{k\in[K]} \overline{\mathbf{X}}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{V}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} \tag{7}$$

such that $\widetilde{\mathbf{X}}_i = x(\alpha_i)$ for all $i \in [N]$.

Lemma 1 ([10], [11]): The Lagrange interpolation polynomial $x(\alpha)$ from (7) combines K shards $\overline{\mathbf{X}}_1,\dots,\overline{\mathbf{X}}_K \in \mathbb{F}_p^{d\times \frac{Nm}{K}}$ of dataset $\mathbf{X} \in \mathbb{F}_p^{d\times Nm}$ along with T random matrices $\mathbf{V}_{K+1},\dots,\mathbf{V}_{K+T} \in \mathbb{F}_p^{d\times \frac{Nm}{K}}$, and ensures that the encoded datasets reveal no information about the true datasets of the honest users even if up to T adversaries collude, which can be formally stated as,

$$I(\{\mathbf{X}_i\}_{i\in\mathcal{H}}; \{\widetilde{\mathbf{X}}_i\}_{i\in\mathcal{T}}, \{[\overline{\mathbf{X}}_i]_j, [\widetilde{\mathbf{X}}_i]_j\}_{i\in\mathcal{H}, j\in\mathcal{T}}, \{[\overline{\mathbf{X}}_i]_j, [\widetilde{\mathbf{X}}_i]_j\}_{i\in\mathcal{T}, j\in[N]}, \{[\mathbf{V}_k]_j\}_{j\in\mathcal{T}, k\in\{K+1,\dots,K+T\}} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) = 0$$

where $\{[\overline{\mathbf{X}}_i]_j, [\widetilde{\mathbf{X}}_i]_j\}_{i\in\mathcal{H}, j\in\mathcal{T}}$ are the secret shares sent from honest users to the adversaries using SSS, $\{[\overline{\mathbf{X}}_i]_j, [\widetilde{\mathbf{X}}_i]_j\}_{i\in\mathcal{T}, j\in[N]}$ denote the secret shares generated by the adversaries, and $\{[\mathbf{V}_k]_j\}_{j\in\mathcal{T}, k\in\{K+1,\dots,K+T\}}$ denote the secret shares of the random matrices $\mathbf{V}_{K+1},\dots,\mathbf{V}_{K+T}$, sent from the crypto-service provider to the adversaries.

In addition to the encoded dataset, each user $i \in [N]$ also holds an encoded model,

$$\widetilde{\mathbf{W}}_{l,i}(t) = \sum_{k\in[K]} \mathbf{W}_l(t) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{Q}_{l,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} \quad \forall l \in [L+1]$$

where $\mathbf{W}_1(t),\dots,\mathbf{W}_{L+1}(t)$ denote the current state of the model parameters at training round t, and $\mathbf{Q}_{l,K+1},\dots,\mathbf{Q}_{l,K+T} \in \mathbb{F}_p^{d_l\times d_{l-1}}$ are T uniformly random matrices. The encoded model $\widetilde{\mathbf{W}}_{l,i}(t)$ can be viewed as an interpolation point of a degree $K+T-1$ Lagrange polynomial,

$$w_l(\alpha) = \sum_{k\in[K]} \mathbf{W}_l(t) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{Q}_{l,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} \ \forall l \in [L+1]$$

where $\widetilde{\mathbf{W}}_{l,i}(t) = w_l(\alpha_i)$.

Using the encoded dataset and model, users then compute the gradient through forward and backward propagation. During forward propagation, each user $i \in [N]$ initially computes the multiplication $\widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i$ from (2) for the first layer $l=1$, which can be viewed as an evaluation point of a degree $2(K+T-1)$ polynomial,

$$f(\alpha) = w_1(\alpha)x(\alpha) = \sum_{k\in[K]} \mathbf{W}_1(t)\overline{\mathbf{X}}_k \prod_{k'\in[K+T]\setminus\{k\}} \Big(\frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}}\Big)^2 + \cdots \tag{8}$$

where the local computation of user i is given by,

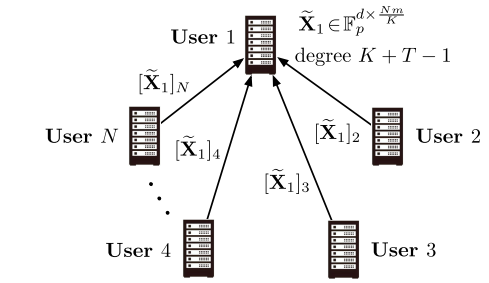
$$f(\alpha_i) = \widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i \in \mathbb{F}_p^{d_1\times \frac{Nm}{K}} \tag{9}$$

and the secret computations for the K shards $\overline{\mathbf{X}}_1,\dots,\overline{\mathbf{X}}_K$ of the dataset $\mathbf{X}$ are given by,

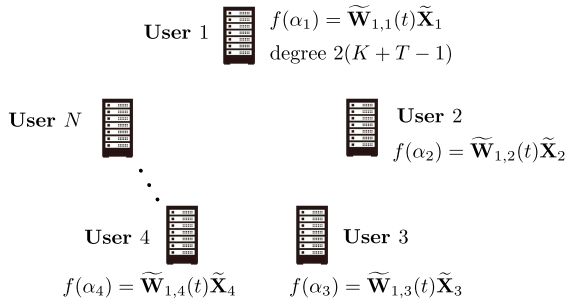
$$f(\beta_k) = \mathbf{W}_1(t)\overline{\mathbf{X}}_k \in \mathbb{F}_p^{d_1\times \frac{Nm}{K}} \quad \forall k \in [K] \tag{10}$$

As illustrated in Fig. 3, the polynomial degree has increased from $K+T-1$ to $deg(f(\alpha)) = 2(K+T-1)$ in (8) due to the multiplication operation. As a result, after L layers, the degree of the resulting polynomial at layer $L+1$, where $\widetilde{\mathbf{W}}_{L+1,i}(t) \times g(\widetilde{\mathbf{W}}_{L,i}(t) \times g(\dots g(\widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i)))$ is the local computation of user i, will be lower bounded by $2^L(K+T-1)$, leading to a degree explosion as illustrated in Fig. 4. Polynomial degree will further increase during backpropagation, and after each training round. After J rounds, the final model $\mathbf{W}_1(J),\dots,\mathbf{W}_{L+1}(J)$ is decoded by using polynomial interpolation. As interpolating any polynomial f requires collecting the computation results from $\deg(f)+1$ users, the final model cannot be recovered if the total number of users is $N-D < J\times 2^L(K+T-1)+1$.

To avoid a degree explosion, conventional approaches reduce the degree after each multiplication operation by utilizing SSS. To reduce the degree of the polynomial in (8) from $2(K+T-1)$ back to $K+T-1$, each user can



(a) Dataset encoding.



(b) Gradient computation.

Fig. 3. Dataset encoding and gradient computation at user 1 using SSS. Initially, each user $j \in [N]$ sends a secret share $[\widetilde{\mathbf{X}}_1]_j$ of an encoded dataset $\widetilde{\mathbf{X}}_1$ to user 1. Upon receiving $\{[\widetilde{\mathbf{X}}_1]_j\}_{j\in[N]}$, user 1 recovers the encoded dataset $\widetilde{\mathbf{X}}_1$. Using the encoded dataset and model, users compute the gradient. The polynomial degree increases with each multiplication operation.

send a secret share $[\widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i]_j$ of its local computation $\widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i$ to user $j \in [N]$, which has a total communication overhead of $O(N^2)$ across the N users. After receiving the shares $\{[\widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i]_j\}_{i\in\mathcal{I}}$ from any set $i \in \mathcal{I}$ of $|\mathcal{I}| = 2(K+T-1)+1$ users, user $j \in [N]$ can decode a secret share $[\mathbf{W}_1(t)\overline{\mathbf{X}}_k]_j$ of the true computation $\mathbf{W}_1(t)\overline{\mathbf{X}}_k$ for each shard $k \in [K]$ using polynomial interpolation,

$$[\mathbf{W}_1(t)\overline{\mathbf{X}}_k]_j = \sum_{i\in\mathcal{I}} [\widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i]_j \prod_{k'\in\mathcal{I}\setminus\{i\}} \frac{\beta_k - \alpha_{k'}}{\alpha_i - \alpha_{k'}} \quad (11)$$

Then, by using the secret shares $\{[\mathbf{W}_1(t)\overline{\mathbf{X}}_k]_j\}_{k\in[K]}$ from (11), users can re-encode the true computations $\{\mathbf{W}_1(t)\overline{\mathbf{X}}_k\}_{k\in[K]}$ using a degree $K+T-1$ (lower-degree) Lagrange interpolation polynomial for the next layer. To do so, each user $j \in [N]$ sends an encoded matrix,

$$[f'(\alpha_i)]_j = \sum_{k\in[K]} [\mathbf{W}_1(t)\overline{\mathbf{X}}_k]_j \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=K+1}^{K+T} [\mathbf{A}_k]_j \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \quad (12)$$

to user $i \in [N]$, where $\mathbf{A}_{K+1}, \ldots, \mathbf{A}_{K+T} \in \mathbb{F}_p^{d_1 \times \frac{Nm}{K}}$ are uniformly random matrices secret shared by the crypto-service provider. The total communication overhead of sending the secret shares from (12) is also quadratic $O(N^2)$ across the N users. After receiving $[f'(\alpha_i)]_j$ from any set $j \in \mathcal{I}'$ of $|\mathcal{I}'| = K+T-1$ users, user i can recover the encoded computation,

$$f'(\alpha_i) = \sum_{k\in[K]} \mathbf{W}_1(t)\overline{\mathbf{X}}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{A}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \quad (13)$$

using polynomial interpolation. $f'(\alpha_i)$ can be viewed as an interpolation point of a degree $K+T-1$ (lower-degree) Lagrange polynomial $f'(\alpha)$ at $\alpha = \alpha_i$, where the true computations corresponding to the K shards $\overline{\mathbf{X}}_1, \ldots, \overline{\mathbf{X}}_K$ of the dataset $\mathbf{X}$ are given by,

$$f'(\beta_k) = \mathbf{W}_1(t)\overline{\mathbf{X}}_k \quad \forall k \in [K] \quad (14)$$

In doing so, the secret computations $\{\mathbf{W}_1(t)\overline{\mathbf{X}}_k\}_{k\in[K]}$ are transferred from a polynomial $f(\alpha)$ of degree $deg(f) = 2(K+T-1)$ to a Lagrange polynomial $f'(\alpha)$ of degree $deg(f') = K+T-1$. During gradient computation, degree reduction should be performed after each layer during forward and backward propagation. A diagram illustrating the gradient computation using the encoded dataset and model is presented in Fig. 5. On the other hand, decoding and re-encoding the K secret computations at each layer $l \in [L+1]$ as in (11) and (13) has a quadratic $O(N^2)$ communication complexity across the N users. As a result, current applications are limited to simpler linear and logistic regression tasks [11], [48], as opposed to neural network training. Our goal is to address this challenge, where we ask,

- Can one train a neural network to solve (4) with linear communication complexity, under the information-theoretic privacy guarantees from (6)?

E. This Work

To address this challenge, in this work we propose CLOVER, a privacy-preserving neural network training framework with linear communication complexity. Our key contribution is a scalable privacy-preserving degree reduction mechanism to enable successive Lagrange coded computations, which we term as Double Lagrange Coding (DLC). Unlike conventional approaches, our degree reduction mechanism incurs only a linear communication overhead $O(N)$ in the number of users, as opposed to quadratic $O(N^2)$.

This mechanism takes as input evaluations of a high degree polynomial $f(\alpha)$ distributed across the N users, where $f(\alpha_i)$ is the coded computation locally evaluated by user i, and $f(\beta_k)$ for $k \in [K]$ denotes the K secret computations, such as the forward propagation operations $f(\beta_k) = \mathbf{W}_1(t)\overline{\mathbf{X}}_k$ from (8) corresponding to the K shards $\overline{\mathbf{X}}_1, \ldots \overline{\mathbf{X}}_K$ of the true dataset $\mathbf{X}$. Our framework then re-encodes the K secret computations $\{f(\beta_k)\}_{k\in[K]}$ by using a lower degree polynomial, without revealing their true values to any user. This is done by decoupling communication into online (data-dependent) and offline (data-agnostic) phases, and offloading the communication-intensive operations to the offline phase, by trading-off quadratic (point-to-point) communications with linear (broadcast). To reduce the degree of the polynomial $f(\alpha)$ from (8), two Lagrange encoded random masks are

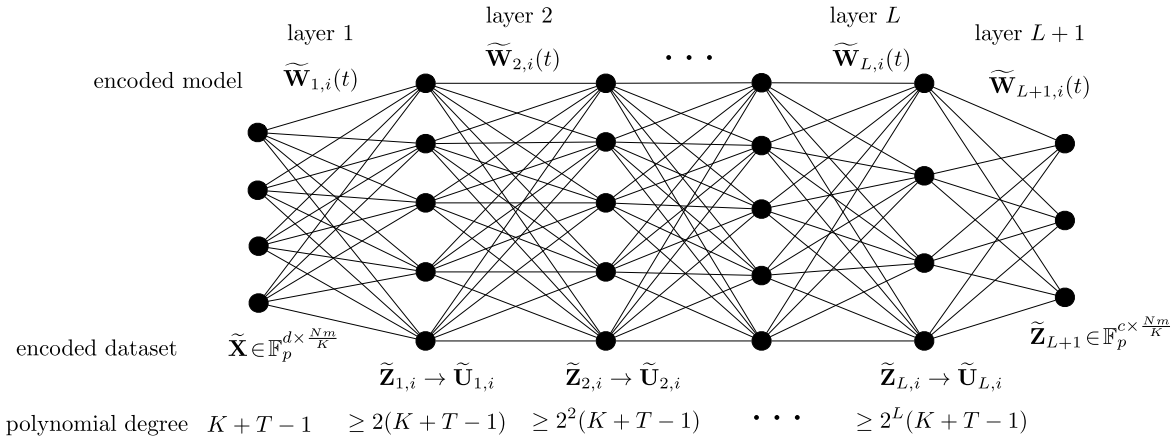


Fig. 4. Illustration of degree explosion during forward propagation. The dataset and model are initially encoded using a degree $K+T-1$ polynomial. The encoded dataset at user $i \in [N]$ is denoted by $\widetilde{\mathbf{X}}_i$, whereas the encoded model parameters connecting layer $l-1$ to layer l is denoted by $\widetilde{\mathbf{W}}_{l,i}(t)$ for $l \in [L+1]$. The degree grows after each multiplication operation, leading to an exponential growth as the number of layers and training rounds increase.

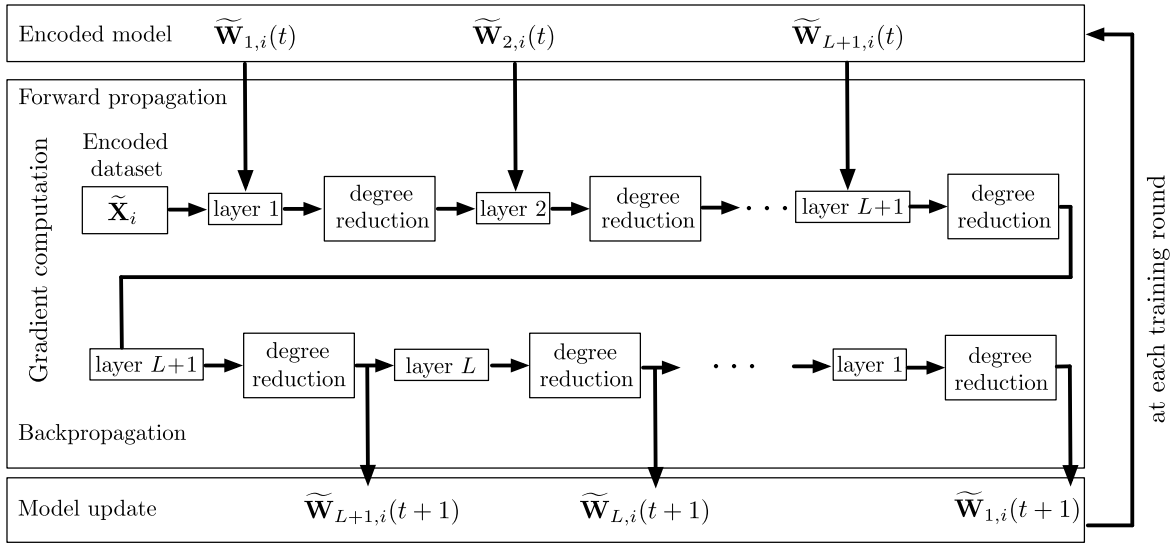


Fig. 5. Diagram of gradient computation and degree reduction steps during training. Each user $i \in [N]$ holds an encoded dataset $\widetilde{\mathbf{X}}_i$, and an encoded model $\widetilde{\mathbf{W}}_{1,i}(t), \ldots, \widetilde{\mathbf{W}}_{L+1,i}(t)$ at training round t. The gradient is then computed through forward and backward propagation of the encoded dataset and model. After each layer $l \in [L+1]$, a degree reduction operation is carried out to reduce the polynomial degree back to $K+T-1$ to avoid a degree explosion.

generated for each user $i \in [N]$ in the offline phase. The first one is the encoded random matrix $\widetilde{\mathbf{R}}_i$,

$$\widetilde{\mathbf{R}}_i = \sum_{k \in [2(K+T-1)+1]} \mathbf{R}_k \prod_{k' \in [2(K+T-1)+1] \setminus \{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{15}$$

which can be viewed as an interpolation point of a degree $2(K+T-1)$ (higher degree) Lagrange polynomial $\phi(\alpha)$ where $\phi(\alpha_i) = \widetilde{\mathbf{R}}_i$, and $\mathbf{R}_1, \ldots, \mathbf{R}_{2(K+T-1)+1} \in \mathbb{F}_p^{d_1 \times \frac{Nm}{K}}$ are $2(K+T-1)+1$ uniformly random matrices. The second one is the encoded matrix $\overline{\mathbf{R}}_i$,

$$\overline{\mathbf{R}}_i = \sum_{k \in [K]} \mathbf{R}_k \prod_{k' \in [K+T] \setminus \{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{A}_k \prod_{k' \in [K+T] \setminus \{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{16}$$

which can be viewed as an interpolation point of a degree $K+T-1$ (lower degree) Lagrange polynomial $\psi(\alpha)$ where $\psi(\alpha_i) = \overline{\mathbf{R}}_i$, and $\mathbf{A}_{K+1}, \ldots, \mathbf{A}_{K+T} \in \mathbb{F}_p^{d_1 \times \frac{Nm}{K}}$ are T uniformly random matrices. As we demonstrate in the following section, the amortized communication complexity for generating $\widetilde{\mathbf{R}}_i$ and $\overline{\mathbf{R}}_i$ can be made linear $O(N)$ in the number of users, by reducing the communication volume to be inversely proportional to the number of users.

In the online phase, the randomness $\{\widetilde{\mathbf{R}}_i\}_{i \in [N]}$ is used to decode a masked version of the true computations $\{\mathbf{W}_1(t)\overline{\mathbf{X}}_k)\}_{k \in [K]}$. To do so, each user broadcasts $f(\alpha_i) - \widetilde{\mathbf{R}}_i$, where the local computation $f(\alpha_i)$ is masked by the randomness $\widetilde{\mathbf{R}}_i$. From (15), $f(\alpha_i) - \widetilde{\mathbf{R}}_i$ can be viewed as an interpolation point of a degree $2(K+T-1)$ polynomial $f(\alpha) - \phi(\alpha)$, where $f(\alpha_i) - \widetilde{\mathbf{R}}_i = f(\alpha_i) - \phi(\alpha_i)$. Then, after receiving $f(\alpha_i) - \widetilde{\mathbf{R}}_i$ from any set of $2(K+T-1)+1$ users, users can decode $f(\beta_k) - \mathbf{R}_k$ using polynomial interpolation, where the true computation $f(\beta_k) = \mathbf{W}_1(t)\overline{\mathbf{X}}_k$ is masked

by the random matrix $\mathbf{R}_k$ for each $k \in [K]$. Then, the second matrix $\overline{\mathbf{R}}_i$ is used to re-encode the true computation $\mathbf{W}_1(t)\overline{\mathbf{X}}_k$ using a lower-degree Lagrange polynomial. This is done by embedding the masked computations $\mathbf{W}_1(t)\overline{\mathbf{X}}_k - \mathbf{R}_k$ in a degree $K+T-1$ polynomial, while simultaneously cancelling the additive randomness $\mathbf{R}_k$ as follows,

$$f'(\alpha_i) = \sum_{k\in[K]} (\mathbf{W}_1(t)\overline{\mathbf{X}}_k - \mathbf{R}_k) \prod_{k\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \overline{\mathbf{R}}_i \tag{17}$$

$$= \sum_{k\in[K]} \mathbf{W}_1(t)\overline{\mathbf{X}}_k \prod_{k\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k\in\{K+1,\ldots,K+T\}} \mathbf{A}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{18}$$

Doing so enables Lagrange coded computations for gradient computations in the subsequent layers, while avoiding a degree explosion due to the increasing number of layers and training rounds. Unlike the conventional approach from (13) using SSS, the communication complexity for re-encoding the true computations using a lower degree Lagrange polynomial is now linear $O(N)$. By leveraging DLC, we then propose a privacy-preserving neural network training framework where DLC is utilized to prevent the degree explosion during gradient calculations. In the following, we describe the individual steps of DLC.

IV. Double Lagrange Coding (DLC)

In this section, we introduce our communication-efficient degree reduction mechanism, DLC, for privacy-preserving iterative polynomial computations. DLC generates two Lagrange interpolation polynomials; a higher degree polynomial to decode a masked version of K secret computations, and a lower degree Lagrange polynomial to re-encode them. The K secret computations from the higher degree polynomial are then transferred to the lower degree polynomial without revealing their true values, while incurring linear communication complexity. In doing so, we leverage MDS matrices for randomness generation, also known as hyperinvertible matrices [4].

Consider a polynomial $f(\cdot)$ of degree $\deg(f) = M$ for some $M \geq K+T-1$, where $f(\beta_1), \ldots, f(\beta_K) \in \mathbb{F}_p^{n_1\times n_2}$ represent the K secret computations, e.g., gradient computations for K data points, for some $n_1, n_2 \in \mathbb{Z}_+$, and $f(\alpha_i)$ is the local coded computation performed by user $i \in [N]$. DLC then generates a new (low degree) Lagrange polynomial $f'(\cdot)$ of degree $K+T-1$, such that $f'(\beta_k) = f(\beta_k)$ are the secret computations for $k \in [K]$, and $f'(\beta_k) \in \mathbb{F}_p^{n_1\times n_2}$ are uniformly random matrices 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. We next describe the individual steps of DLC, which is expressed in the sequel as,

$$f'(\alpha_1), \ldots, f'(\alpha_N) \leftarrow DLC(f(\alpha_1), \ldots, f(\alpha_N), M) \tag{19}$$

DLC consists of offline (data-agnostic) and online (data-dependent) phases. The offline phase is independent from the datasets, hence can be carried out offline when the network load is low. The online phase depends on the datasets, and is carried out after training starts.

(Offline) In the offline phase, users first agree on $M-K+1$ distinct public parameters $\beta_{K+1}, \ldots, \beta_{M+1} \in \mathbb{F}_p$ such that $\{\beta_1, \ldots, \beta_K\} \cap \{\beta_{K+1}, \ldots, \beta_{M+1}\} = \emptyset$. Each user $i \in [N]$ then generates $M+1$ uniformly random matrices $\mathbf{R}_{i,1}, \ldots, \mathbf{R}_{i,M+1}$ of size $\frac{n_1}{N-T} \times n_2$ from $\mathbb{F}_p$, and forms a Lagrange interpolation polynomial of degree M,

$$\phi_i(\alpha) = \sum_{k\in[M+1]} \mathbf{R}_{i,k} \prod_{k'\in[M+1]\setminus\{k\}} \frac{\alpha - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{20}$$

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

$$\widetilde{\mathbf{R}}_{i,j} = \phi_i(\alpha_j) \in \mathbb{F}_p^{\frac{n_1}{N-T}\times n_2} \tag{21}$$

to user $j \in [N]$. In addition to (20), user i also creates a second (lower-degree) Lagrange polynomial with degree $K+T-1$,

$$\psi_i(\alpha) = \sum_{k\in[K]} \mathbf{R}_{i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{A}_{i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{22}$$

where $\mathbf{A}_{i,k} \in \mathbb{F}_p^{\frac{n_1}{N-T}\times n_2}$ are generated uniformly random for $k \in \{K+1, \ldots, K+T\}$. Then, user i sends an encoded matrix,

$$\overline{\mathbf{R}}_{i,j} = \psi_i(\alpha_j) \in \mathbb{F}_p^{\frac{n_1}{N-T}\times n_2} \tag{23}$$

to user $j \in [N]$. After receiving $\{\widetilde{\mathbf{R}}_{j,i}, \overline{\mathbf{R}}_{j,i}\}_{j\in[N]}$, user $i \in [N]$ combines them to generate two higher-dimensional encoded matrices $\widetilde{\mathbf{R}}_i, \overline{\mathbf{R}}_i \in \mathbb{F}_p^{n_1\times n_2}$,

$$\widetilde{\mathbf{R}}_i = \begin{bmatrix} \sum_{j\in[N]} \lambda_1^{j-1}\widetilde{\mathbf{R}}_{j,i} \\ \vdots \\ \sum_{j\in[N]} \lambda_{N-T}^{j-1}\widetilde{\mathbf{R}}_{j,i} \end{bmatrix} = \sum_{k\in[M+1]} \mathbf{R}_k \prod_{k'\in[M+1]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{24}$$

and

$$\overline{\mathbf{R}}_i = \begin{bmatrix} \sum_{j\in[N]} \lambda_1^{j-1}\overline{\mathbf{R}}_{j,i} \\ \vdots \\ \sum_{j\in[N]} \lambda_{N-T}^{j-1}\overline{\mathbf{R}}_{j,i} \end{bmatrix} = \sum_{k\in[K]} \mathbf{R}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{A}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{25}$$

where

$$\mathbf{R}_k = \begin{bmatrix} \sum_{j\in[N]} \lambda_1^{j-1}\mathbf{R}_{j,k} \\ \vdots \\ \sum_{j\in[N]} \lambda_{N-T}^{j-1}\mathbf{R}_{j,k} \end{bmatrix} \quad \forall k \in [M+1] \tag{26}$$

and

$$\mathbf{A}_k = \begin{bmatrix} \sum_{j\in[N]} \lambda_1^{j-1}\mathbf{A}_{j,k} \\ \vdots \\ \sum_{j\in[N]} \lambda_{N-T}^{j-1}\mathbf{A}_{j,k} \end{bmatrix} \quad \forall k \in \{K+1,\ldots,K+T\} \tag{27}$$

In doing so, the goal is to generate high-dimensional shared coded randomness using the low-dimensional random matrices generated locally by each user. The dimension of the encoded random matrices $\{\widetilde{\mathbf{R}}_{i,j}, \overline{\mathbf{R}}_{i,j}\}_{j\in[N]}$ locally generated by each user $i \in [N]$ has size $\frac{n_1}{N-T} \times n_2$, whereas the size of the final encoded randomness $\widetilde{\mathbf{R}}_i, \overline{\mathbf{R}}_i$ from (24) and (25) have size $n_1 \times n_2$.

(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 polynomial $\varphi(\alpha) = f(\alpha) - \phi(\alpha)$ where,

$$\phi(\alpha) = \sum_{k\in[M+1]} \mathbf{R}_k \prod_{k'\in[M+1]\setminus\{k\}} \frac{\alpha - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{28}$$

such that,

$$\varphi(\alpha_i) = f(\alpha_i) - \phi(\alpha_i) = f(\alpha_i) - \widetilde{\mathbf{R}}_i \qquad \forall i \in [N], \tag{29}$$

corresponds to the local computation of user i masked by $\widetilde{\mathbf{R}}_i$ from (24), and

$$\varphi(\beta_k) = f(\beta_k) - \phi(\beta_k) = f(\beta_k) - \mathbf{R}_k \qquad \forall k \in [K] \tag{30}$$

corresponds to the secret computation $f(\beta_k)$ masked by the random additive mask $\mathbf{R}_k = \phi(\beta_k)$ from (26) to hide its true value. After receiving $\varphi(\alpha_j)$ from any set $j \in \mathcal{I}$ of at least $|\mathcal{I}| \geq M+1$ users, each user can decode $\varphi(\beta_k) = f(\beta_k) - \phi(\beta_k) = f(\beta_k) - \mathbf{R}_k$ for all $k \in [K]$ through polynomial interpolation,

$$\varphi(\beta_k) = \sum_{j\in\mathcal{I}} \varphi(\alpha_j) \prod_{k'\in\mathcal{I}\setminus\{j\}} \frac{\beta_k - \alpha_{k'}}{\alpha_j - \alpha_{k'}} \quad \forall k \in [K] \tag{31}$$

Finally, each user $i \in [N]$ re-encodes $\{f(\beta_k)\}_{k\in[K]}$ as follows,

$$f'(\alpha_i) = \sum_{k\in[K]} \varphi(\beta_k) \prod_{k\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \overline{\mathbf{R}}_i \tag{32}$$

$$= \sum_{k\in[K]} (f(\beta_k) - \mathbf{R}_k) \prod_{k\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \overline{\mathbf{R}}_i \tag{33}$$

$$= \sum_{k\in[K]} f(\beta_k) \prod_{k\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k\in\{K+1,\ldots,K+T\}} \mathbf{A}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{34}$$

which simultaneously cancels the additive randomness $\{\mathbf{R}_k\}_{k\in[K]}$, and embeds the desired computations to a low degree (degree $K+T-1$) Lagrange polynomial, along with T random matrices $\{\mathbf{A}_k\}_{k\in\{K+1,\ldots,K+T\}}$. For neural network training, which is the main focus of our work, DLC is utilized to reduce the degree of the polynomials embedding the true gradient computations, which are computed using the Lagrange coded datasets. On the other hand, our degree reduction mechanism can be leveraged for any iterative algorithm building on polynomial computations, beyond machine learning. In the following, we demonstrate a motivating example for DLC.

A. Motivating Example for DLC

We next present a motivating example for DLC. Consider $N = 5$ users, with $T = 1$, $D = 0$, and the parallelization degree $K = 2$. Each user has $m = 2$ data samples with $d = 2$ features. We then consider degree reduction for the polynomial $f(\alpha)$ from (8) during forward propagation of a model $\mathbf{W}_1(t) \in \mathbb{F}_p^{4\times 2}$ with $d_1 = 4$ neurons at layer $l = 1$, along with a dataset $\mathbf{X} \in \mathbb{F}_p^{2\times 10}$ partitioned into $K = 2$ equal-sized shards $\overline{\mathbf{X}}_1, \overline{\mathbf{X}}_2 \in \mathbb{F}_p^{2\times 5}$. Each user $i \in [5]$ holds an encoded dataset $\widetilde{\mathbf{X}}_i \in \mathbb{F}_p^{2\times 5}$ and model $\widetilde{\mathbf{W}}_{1,i}(t) \in \mathbb{F}_p^{4\times 2}$. From (8), the degree of $f(\alpha)$ is $M = 2(K+T-1) = 4$, where $f(\alpha_i) = \widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i \in \mathbb{F}_p^{4\times 5}$ is the local computation of user $i \in [5]$, and $f(\beta_1) = \mathbf{W}_1(t)\overline{\mathbf{X}}_1$, $f(\beta_2) = \mathbf{W}_1(t)\overline{\mathbf{X}}_2 \in \mathbb{F}_p^{4\times 5}$ denote the secret computations for the two shards $\overline{\mathbf{X}}_1, \overline{\mathbf{X}}_2$, respectively. Then, degree reduction $f'(\alpha_1), \ldots, f'(\alpha_5) = DLC(f(\alpha_1), \ldots, f(\alpha_5), 5)$ consists of the following offline and online phases.

Offline Phase: In the offline phase, each user $i \in [5]$ initially generates $M+1 = 5$ uniformly random matrices $\mathbf{R}_{i,1}, \ldots, \mathbf{R}_{i,5} \in \mathbb{F}_p^{1\times 5}$, and sends an encoded matrix,

$$\widetilde{\mathbf{R}}_{i,j} = \mathbf{R}_{i,1} \prod_{k'\in[5]\setminus\{1\}} \frac{\alpha_j - \beta_{k'}}{\beta_1 - \beta_{k'}} + \mathbf{R}_{i,2} \prod_{k'\in[5]\setminus\{2\}} \frac{\alpha_j - \beta_{k'}}{\beta_2 - \beta_{k'}} + \ldots + \mathbf{R}_{i,5} \prod_{k'\in[5]\setminus\{5\}} \frac{\alpha_j - \beta_{k'}}{\beta_5 - \beta_{k'}} \tag{35}$$

to user $j \in [5]$. In addition, user $i \in [5]$ generates a uniformly random matrix $\mathbf{A}_{i,3} \in \mathbb{F}_p^{1\times 5}$, and sends an encoded matrix

$$\overline{\mathbf{R}}_{i,j} = \mathbf{R}_{i,1} \prod_{k'\in[5]\setminus\{1\}} \frac{\alpha_j - \beta_{k'}}{\beta_1 - \beta_{k'}} + \mathbf{R}_{i,2} \prod_{k'\in[5]\setminus\{2\}} \frac{\alpha_j - \beta_{k'}}{\beta_2 - \beta_{k'}} + \mathbf{A}_{i,3} \prod_{k'\in[5]\setminus\{3\}} \frac{\alpha_j - \beta_{k'}}{\beta_3 - \beta_{k'}} \tag{36}$$

to user $j \in [5]$. Then, each user $i \in [5]$ combines the received matrices $\widetilde{\mathbf{R}}_{1,i}, \ldots, \widetilde{\mathbf{R}}_{5,i} \in \mathbb{F}_p^{1\times 5}$ and $\overline{\mathbf{R}}_{1,i}, \ldots, \overline{\mathbf{R}}_{5,i} \in \mathbb{F}_p^{1\times 5}$ to generate two higher dimensional coded random matrices, $\widetilde{\mathbf{R}}_i \in \mathbb{F}_p^{4\times 5}$,

$$\widetilde{\mathbf{R}}_i = \begin{bmatrix} \sum_{j\in[5]} \lambda_1^{j-1}\widetilde{\mathbf{R}}_{j,i} \\ \vdots \\ \sum_{j\in[5]} \lambda_4^{j-1}\widetilde{\mathbf{R}}_{j,i} \end{bmatrix} \tag{37}$$

$$= \underbrace{\begin{bmatrix} \sum_{j\in[5]} \lambda_1^{j-1}\mathbf{R}_{j,1} \\ \vdots \\ \sum_{j\in[5]} \lambda_4^{j-1}\mathbf{R}_{j,1} \end{bmatrix}}_{\mathbf{R}_1} \prod_{k'\in[5]\setminus\{1\}} \frac{\alpha_i-\beta_{k'}}{\beta_1-\beta_{k'}} + \ldots + \underbrace{\begin{bmatrix} \sum_{j\in[5]} \lambda_1^{j-1}\mathbf{R}_{j,5} \\ \vdots \\ \sum_{j\in[5]} \lambda_4^{j-1}\mathbf{R}_{j,5} \end{bmatrix}}_{\mathbf{R}_5} \prod_{k'\in[5]\setminus\{5\}} \frac{\alpha_i-\beta_{k'}}{\beta_5-\beta_{k'}} \tag{38}$$

$$= \mathbf{R}_1 \prod_{k'\in[5]\setminus\{1\}} \frac{\alpha_i-\beta_{k'}}{\beta_1-\beta_{k'}} + \mathbf{R}_2 \prod_{k'\in[5]\setminus\{2\}} \frac{\alpha_i-\beta_{k'}}{\beta_2-\beta_{k'}} + \ldots + \mathbf{R}_5 \prod_{k'\in[5]\setminus\{5\}} \frac{\alpha_i-\beta_{k'}}{\beta_5-\beta_{k'}} \tag{39}$$

and $\overline{\mathbf{R}}_i \in \mathbb{F}_p^{4\times 5}$,

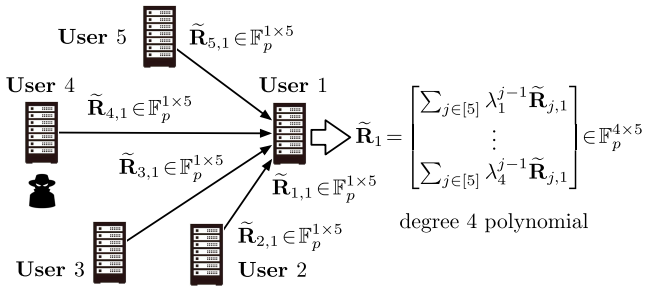
$$\overline{\mathbf{R}}_i = \begin{bmatrix} \sum_{j\in[5]} \lambda_1^{j-1}\overline{\mathbf{R}}_{j,i} \\ \vdots \\ \sum_{j\in[5]} \lambda_4^{j-1}\overline{\mathbf{R}}_{j,i} \end{bmatrix} \tag{40}$$

$$= \sum_{k\in[2]} \underbrace{\begin{bmatrix} \sum_{j\in[5]} \lambda_1^{j-1}\mathbf{R}_{j,k} \\ \vdots \\ \sum_{j\in[5]} \lambda_4^{j-1}\mathbf{R}_{j,k} \end{bmatrix}}_{\mathbf{R}_k} \prod_{k'\in[3]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} + \underbrace{\begin{bmatrix} \sum_{j\in[5]} \lambda_1^{j-1}\mathbf{A}_{j,3} \\ \vdots \\ \sum_{j\in[5]} \lambda_4^{j-1}\mathbf{A}_{j,3} \end{bmatrix}}_{\mathbf{A}_3} \prod_{k'\in[3]\setminus\{3\}} \frac{\alpha_i-\beta_{k'}}{\beta_3-\beta_{k'}} \tag{41}$$

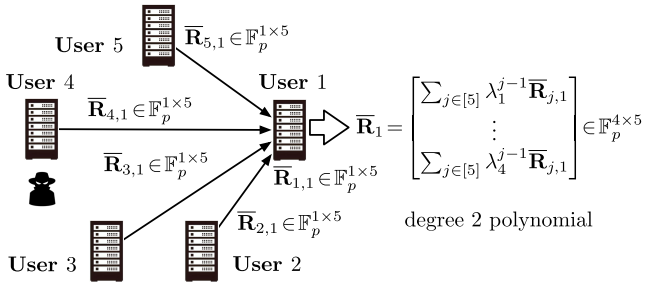
$$= \mathbf{R}_1 \prod_{k'\in[3]\setminus\{1\}} \frac{\alpha_i-\beta_{k'}}{\beta_1-\beta_{k'}} + \mathbf{R}_2 \prod_{k'\in[3]\setminus\{2\}} \frac{\alpha_i-\beta_{k'}}{\beta_2-\beta_{k'}} + \mathbf{A}_3 \prod_{k'\in[3]\setminus\{3\}} \frac{\alpha_i-\beta_{k'}}{\beta_3-\beta_{k'}} \tag{42}$$

Hence, in the offline phase each user $i \in [5]$ sends two encoded random matrices $\widetilde{\mathbf{R}}_{i,j}, \overline{\mathbf{R}}_{i,j}$ of dimension 1×5. The first matrix $\widetilde{\mathbf{R}}_{i,j}$ is generated using a degree $2(K+T-1)=4$ (higher-degree) Lagrange interpolation polynomial, whereas the second matrix $\overline{\mathbf{R}}_{i,j}$ is generated using a degree $K+T-1=2$ (lower-degree) Lagrange polynomial. Upon receiving the lower-dimensional coded matrices $\{\widetilde{\mathbf{R}}_{j,i}, \overline{\mathbf{R}}_{j,i}\}_{j\in[5]}$, user i then generates two higher-dimensional coded matrices $\widetilde{\mathbf{R}}_i, \overline{\mathbf{R}}_i$, each of dimension 4×5, to be used later in the online phase for degree reduction. Fig. 6 illustrates the offline phase of DLC and the generation of the encoded randomness for user 1.

Online Phase: In the online phase, each user $i \in [5]$ initially broadcasts $\widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i - \widetilde{\mathbf{R}}_i$, where $\widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i \in \mathbb{F}_p^{4\times 5}$ is the local computation of user i from (8), which can be viewed as an interpolation point of the degree $2(K+T-1)=4$ polynomial $f(\alpha)$ from (8) such that $f(\alpha_i) = \widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i$, whereas



(a) Generation of the high degree polynomial.



(b) Generation of the low degree polynomial.

Fig. 6. Illustration of Double Lagrange Coding (DLC) offline phase, with $N=5$ and $T=1$. Upon receiving two lower-dimensional encoded random matrices $\widetilde{\mathbf{R}}_{j,1}, \overline{\mathbf{R}}_{j,1} \in \mathbb{F}_p^{1\times 5}$ from users $j \in [5]$, user 1 generates two higher-dimensional encoded random matrices $\widetilde{\mathbf{R}}_1, \overline{\mathbf{R}}_1 \in \mathbb{F}_p^{4\times 5}$, which will be used later in the online phase for degree reduction.

$f(\beta_k) = \mathbf{W}_1(t)\overline{\mathbf{X}}_k \in \mathbb{F}_p^{4\times 5}$ denote the secret computations for the two shards $\overline{\mathbf{X}}_k$ for $k \in [2]$. Similarly, $\widetilde{\mathbf{R}}_i$ is an interpolation point of the degree $2(K+T-1)=4$ polynomial $\phi(\alpha)$ from (28) such that $\phi(\alpha_i) = \widetilde{\mathbf{R}}_i \in \mathbb{F}_p^{4\times 5}$, whereas $\phi(\beta_k) = \mathbf{R}_k \in \mathbb{F}_p^{4\times 5}$ for $k \in [2]$. Then, after receiving $\widetilde{\mathbf{W}}_{1,i}(t)\widetilde{\mathbf{X}}_i - \widetilde{\mathbf{R}}_i$ from $2(K+T-1)+1=5$ users, users can recover the polynomial $f(\beta_k)-\phi(\beta_k)$ using polynomial interpolation,

$$\mathbf{W}_1(t)\overline{\mathbf{X}}_k - \mathbf{R}_k = \sum_{j\in[5]} \widetilde{\mathbf{W}}_{1,j}(t)\widetilde{\mathbf{X}}_j \prod_{k'\in[5]\setminus\{j\}} \frac{\beta_k-\alpha_{k'}}{\alpha_j-\alpha_{k'}} \text{ for } k\in[2] \tag{43}$$

where the true computation $\mathbf{W}_1(t)\overline{\mathbf{X}}_k$ is hidden by the additive random mask $\mathbf{R}_k$. Finally, user $i \in [5]$ re-encodes the secret computation $\mathbf{W}_1(t)\overline{\mathbf{X}}_k$ in a Lagrange interpolation polynomial of degree $K+T-1=2$ as follows,

$$f'(\alpha_i) = \sum_{k\in[2]} (\mathbf{W}_1(t)\overline{\mathbf{X}}_k - \mathbf{R}_k) \prod_{k'\in[3]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} + \overline{\mathbf{R}}_i \tag{44}$$

$$= \sum_{k\in[2]} \mathbf{W}_1(t)\overline{\mathbf{X}}_k \prod_{k'\in[3]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} - \sum_{k\in[2]} \mathbf{R}_k \prod_{k'\in[3]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} \tag{45}$$

$$+ \sum_{k\in[2]} \mathbf{R}_k \prod_{k'\in[3]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} + \mathbf{A}_3 \prod_{k'\in[3]\setminus\{3\}} \frac{\alpha_i-\beta_{k'}}{\beta_3-\beta_{k'}} \tag{46}$$

$$= \sum_{k\in[2]} \mathbf{W}_1(t)\overline{\mathbf{X}}_k \prod_{k'\in[3]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} + \mathbf{A}_3 \prod_{k'\in[3]\setminus\{3\}} \frac{\alpha_i-\beta_{k'}}{\beta_3-\beta_{k'}} \tag{47}$$

where the additive masks $\mathbf{R}_1, \mathbf{R}_2$ cancel out. As a result, each user $i \in [5]$ learns an encoded matrix $f'(\alpha_i)$ where the true computations $\mathbf{W}_1(t)\overline{\mathbf{X}}_1, \mathbf{W}_1(t)\overline{\mathbf{X}}_2$ are now embedded in a lower-degree Lagrange polynomial $f'(\alpha)$, which is of degree $K+T-1=2$. Fig. 7 illustrates the online phase of DLC and the degree reduction process for user 1. In the following, we introduce our privacy-preserving neural network training framework CLOVER, which leverages DLC to avoid degree explosion during gradient computations at successive layers.

V. Privacy-Preserving Neural Network Training With CLOVER

In this section, we present our privacy-preserving neural network training framework CLOVER, which utilizes DLC for degree reduction. CLOVER consists of five key components: *1) Dataset Encoding, 2) Label Encoding, 3) Model Initialization, 4) Gradient Computation, 5) Model Update.* Initially, users encode their datasets and labels using a Lagrange interpolation polynomial of degree $K+T-1$. At the end, each user $i \in [N]$ learns an encoded dataset $\widetilde{\mathbf{X}}_i$ and encoded labels $\widetilde{\mathbf{Y}}_i$. The encoding process has two key features. First, it distributes the computation load across the N users, such that the computation load per-user scales with respect to $1/K$ for the intensive gradient computations during training. Next, it ensures the information-theoretic privacy of the sensitive datasets and labels against up to T colluding adversarial users. In an offline phase prior to training, the model is initialized randomly, but without revealing its true value to any user, and encoded using a Lagrange interpolation polynomial. Each user $i \in [N]$ then learns an encoded model $\widetilde{\mathbf{W}}_{1,i}(0), \ldots, \widetilde{\mathbf{W}}_{L+1,i}(0)$, but without learning the true model. Model encoding ensures the privacy of the intermediate training computations. At each training round $t \in \{0, \ldots, J-1\}$, users leverage DLC to compute the gradient using the encoded dataset $\widetilde{\mathbf{X}}_i$, labels $\widetilde{\mathbf{Y}}_i$, and model $\widetilde{\mathbf{W}}_{1,i}(t), \ldots, \widetilde{\mathbf{W}}_{L+1,i}(t)$. At the end, user i learns an encoded gradient $\widetilde{\mathbf{G}}_{1,i}(t), \ldots, \widetilde{\mathbf{G}}_{L+1,i}(t)$, using which the user updates the model for the next training round.

In the following, we describe the details of the individual components. For clarity of presentation, we present the offline and online phases sequentially, to demonstrate how the variables generated in the offline phase are utilized in the online phase. We note, however, that in practice all offline phases can be fully carried out in parallel in advance; the variables generated in the offline phases are independent and do not depend on the previous online/offline phases.

A. Dataset Encoding

Initially, users encode their datasets using locally generated randomness. The goal of dataset encoding is two-fold: 1) hide the dataset against adversaries, 2) reduce the size of data processed during training. In particular, after dataset encoding, 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, thus reducing the computation load per-user which increases the parallelization gain and speeds up training.

For dataset encoding, users first agree on $N+K+T$ distinct public parameters $\{\alpha_j\}_{j\in[N]}, \{\beta_j\}_{j\in[K+T]}$ from $\mathbb{F}_p$. Each user $i \in [N]$ then partitions its local dataset into K equal-sized shards $\mathbf{X}_i = \begin{bmatrix}\mathbf{X}_{i,1} & \cdots & \mathbf{X}_{i,K}\end{bmatrix}$ and sends an encoded matrix,

$$\widetilde{\mathbf{X}}_{i,j} = \sum_{k\in[K]} \mathbf{X}_{i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{V}_{i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j-\beta_{k'}}{\beta_k-\beta_{k'}} \tag{48}$$

to each user $j \in [N]$, where $\mathbf{V}_{i,K+1}, \ldots, \mathbf{V}_{i,K+T} \in \mathbb{F}_p^{d\times \frac{m}{K}}$ are generated independently and uniformly at random. By concatenating the received coded matrices $\{\widetilde{\mathbf{X}}_{j,i}\}_{j\in[N]}$, each user i constructs an encoded dataset,

$$\widetilde{\mathbf{X}}_i = \begin{bmatrix}\widetilde{\mathbf{X}}_{1,i} & \cdots & \widetilde{\mathbf{X}}_{N,i}\end{bmatrix} \in \mathbb{F}_p^{d\times \frac{Nm}{K}} \tag{49}$$

which can be viewed as an interpolation point of a degree $K+T-1$ Lagrange polynomial,

$$x(\alpha) = \sum_{k\in[K]} \begin{bmatrix}\mathbf{X}_{1,k} & \cdots & \mathbf{X}_{N,k}\end{bmatrix} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=K+1}^{K+T} \begin{bmatrix}\mathbf{V}_{1,k} & \cdots & \mathbf{V}_{N,k}\end{bmatrix} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} \tag{50}$$

$$= \sum_{k\in[K]} \overline{\mathbf{X}}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{V}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} \tag{51}$$

where $\widetilde{\mathbf{X}}_i = x(\alpha_i)$, and $\overline{\mathbf{X}}_k \triangleq \begin{bmatrix}\mathbf{X}_{1,k} & \cdots & \mathbf{X}_{N,k}\end{bmatrix} = x(\beta_k) \in \mathbb{F}_p^{d\times \frac{Nm}{K}}$ is a matrix whose size is $(1/K)^{th}$ of the true dataset $\mathbf{X}$. As a result, each user receives an encoded version of the datasets of other users. The T random matrices $\mathbf{V}_k \triangleq \begin{bmatrix}\mathbf{V}_{1,k} & \cdots & \mathbf{V}_{N,k}\end{bmatrix}$ for $k \in \{K+1, \ldots, K+T\}$ hide the true dataset against up to T adversaries. As the encoding process depends on the local datasets of the users, dataset encoding is carried out online during training.

B. Label Encoding

In addition to encoding the datasets, users also encode the labels using a Lagrange interpolation polynomial. To this end, each user $i \in [N]$ first partitions its local labels into K equal-sized shards $\mathbf{Y}_i = \begin{bmatrix}\mathbf{Y}_{i,1} & \cdots & \mathbf{Y}_{i,K}\end{bmatrix}$, and sends an encoded matrix,

$$\widetilde{\mathbf{Y}}_{i,j} = \sum_{k\in[K]} \mathbf{Y}_{i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{N}_{i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j-\beta_{k'}}{\beta_k-\beta_{k'}} \tag{52}$$

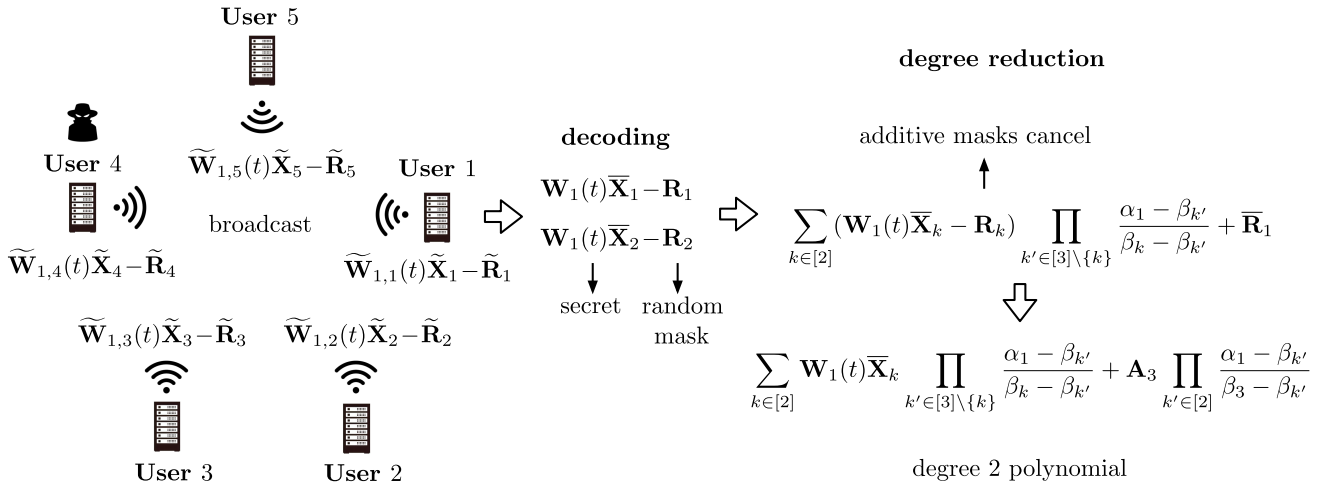


Fig. 7. Illustration of DLC for the online phase. User $i \in [5]$ initially holds the coded computation $\widetilde{\mathbf{W}}_{1,i}\widetilde{\mathbf{X}}_i$, which corresponds to an evaluation point of a degree 4 polynomial. The goal is to reduce the polynomial degree to $K+T-1=2$, without revealing the secret computations $\mathbf{W}_1\overline{\mathbf{X}}_1$, $\mathbf{W}_1\overline{\mathbf{X}}_2$ corresponding to the two shards $\overline{\mathbf{X}}_1, \overline{\mathbf{X}}_2$ of the true dataset $\mathbf{X}$. To do so, each user $i \in [5]$ initially broadcasts a masked computation $\widetilde{\mathbf{W}}_{1,i}\widetilde{\mathbf{X}}_i - \widetilde{\mathbf{R}}_i$ to the other users. Upon receiving $\{\widetilde{\mathbf{W}}_{1,i}\widetilde{\mathbf{X}}_i - \widetilde{\mathbf{R}}_i\}_{i\in[5]}$, users can decode $\mathbf{W}_1\overline{\mathbf{X}}_1 - \mathbf{R}_1$, $\mathbf{W}_1\overline{\mathbf{X}}_2 - \mathbf{R}_2$, where the secret computations are masked by the additive random masks $\mathbf{R}_1$ and $\mathbf{R}_2$, respectively. User $i \in [5]$ then re-encodes the secret computations by using $\overline{\mathbf{R}}_i$, which cancels the additive masks $\mathbf{R}_1, \mathbf{R}_2$ and generates a lower-degree Lagrange polynomial with degree 2.

to each user $j \in [N]$, where $\mathbf{N}_{i,K+1}, \ldots, \mathbf{N}_{i,K+T} \in \mathbb{F}_p^{c\times \frac{m}{K}}$ are generated independently and uniformly at random. By concatenating the received matrices $\{\widetilde{\mathbf{Y}}_{j,i}\}_{j\in[N]}$, each user i obtains the encoded labels,

$$\widetilde{\mathbf{Y}}_i = \begin{bmatrix}\widetilde{\mathbf{Y}}_{1,i} & \cdots & \widetilde{\mathbf{Y}}_{N,i}\end{bmatrix} \in \mathbb{F}_p^{c\times \frac{Nm}{K}} \tag{53}$$

which can be viewed as an interpolation point of a degree $K+T-1$ Lagrange polynomial,

$$\begin{aligned} y(\alpha) &= \sum_{k\in[K]} \begin{bmatrix}\mathbf{Y}_{1,k} & \cdots & \mathbf{Y}_{N,k}\end{bmatrix} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} \\ &\quad + \sum_{k=K+1}^{K+T} \begin{bmatrix}\mathbf{N}_{1,k} & \cdots & \mathbf{N}_{N,k}\end{bmatrix} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} \end{aligned} \tag{54}$$

$$\begin{aligned} &= \sum_{k\in[K]} \overline{\mathbf{Y}}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} \\ &\quad + \sum_{k=K+1}^{K+T} \mathbf{N}_k \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} \end{aligned} \tag{55}$$

where $\widetilde{\mathbf{Y}}_i = y(\alpha_i)$, and $\overline{\mathbf{Y}}_k \triangleq \begin{bmatrix}\mathbf{Y}_{1,k} & \cdots & \mathbf{Y}_{N,k}\end{bmatrix} = y(\beta_k) \in \mathbb{F}_p^{d\times \frac{Nm}{K}}$ is a matrix whose size is $(1/K)^{th}$ of the true dataset $\mathbf{Y}$. As a result, each user receives an encoded version of the labels of other users. In doing so, the T random matrices $\mathbf{N}_k \triangleq \begin{bmatrix}\mathbf{N}_{1,k} & \cdots & \mathbf{N}_{N,k}\end{bmatrix}$ for $k \in \{K+1, \ldots, K+T\}$ hide the true labels against up to T adversaries. As the encoding process depends on the local labels of the users, label encoding is also carried out online during training.

C. Model Initialization

To preserve the privacy of intermediate training computations, the model $\mathbf{W}_1(0), \ldots, \mathbf{W}_{L+1}(0)$ at round $t=0$ should be initialized without revealing their true value to the users, even if up to T users collude. To do so, users first agree on $N-T$ distinct public parameters $\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{N-T}$ from $\mathbb{F}_p$. Then, for each layer $l \in [L+1]$, user $i \in [N]$ generates $T+1$ matrices $\mathbf{W}_{l,i}(0), \mathbf{Q}_{l,i,K+1}, \ldots, \mathbf{Q}_{l,i,K+T} \in \mathbb{F}_p^{\frac{d_l}{N-T}\times d_{l-1}}$ independently and uniformly at random, and then sends an encoded matrix,

$$\begin{aligned} \widetilde{\mathbf{W}}_{l,i,j}(0) &= \sum_{k\in[K]} \mathbf{W}_{l,i}(0) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j-\beta_{k'}}{\beta_k-\beta_{k'}} \\ &\quad + \sum_{k\in\{K+1,\ldots,K+T\}} \mathbf{Q}_{l,i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j-\beta_{k'}}{\beta_k-\beta_{k'}} \end{aligned} \tag{56}$$

to user $j \in [N]$. After receiving $\{\widetilde{\mathbf{W}}_{l,j,i}(0)\}_{j\in[N]}$, user $i \in [N]$ generates a higher-dimensional encoded model $\widetilde{\mathbf{W}}_{l,i}(0) \in \mathbb{F}_p^{d_l\times d_{l-1}}$,

$$\widetilde{\mathbf{W}}_{l,i}(0) = \begin{bmatrix} \sum_{j\in[N]} \boldsymbol{\lambda}_1^{j-1} \widetilde{\mathbf{W}}_{l,j,i}(0) \\ \vdots \\ \sum_{j\in[N]} \boldsymbol{\lambda}_{N-T}^{j-1} \widetilde{\mathbf{W}}_{l,j,i}(0) \end{bmatrix} \tag{57}$$

$$\begin{aligned} &= \sum_{k\in[K]} \mathbf{W}_l(0) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} \\ &\quad + \sum_{k\in\{K+1,\ldots,K+T\}} \mathbf{Q}_{l,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} \end{aligned} \tag{58}$$

where the true model initialized at layer l is given by,

$$\mathbf{W}_l(0) = \begin{bmatrix} \sum_{j\in[N]} \boldsymbol{\lambda}_1^{j-1} \mathbf{W}_{l,j}(0) \\ \vdots \\ \sum_{j\in[N]} \boldsymbol{\lambda}_{N-T}^{j-1} \mathbf{W}_{l,j}(0) \end{bmatrix} \in \mathbb{F}_p^{d_l\times d_{l-1}} \tag{59}$$

whose true value is masked by the T random matrices,

$$\mathbf{Q}_{l,k} = \begin{bmatrix} \sum_{j\in[N]} \lambda_1^{j-1}\mathbf{Q}_{l,j,k} \\ \vdots \\ \sum_{j\in[N]} \lambda_{N-T}^{j-1}\mathbf{Q}_{l,j,k} \end{bmatrix} \in \mathbb{F}_p^{d_l\times d_{l-1}} \text{ for } k\in\{K+1,\dots,K+T\} \tag{60}$$

As such, to generate a random coded matrix of size $d_l \times d_{l-1}$, each user sends a matrix of size $\frac{d_l}{N-T}\times d_{l-1}$. The final encoded matrix is then generated by combining the lower-dimensional coded matrices. The encoded model $\widetilde{\mathbf{W}}_{l,i}(t)$ can be viewed as an interpolation point of a degree $K+T-1$ polynomial,

$$w_l(\alpha) = \sum_{k\in[K]} \mathbf{W}_l(t) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{Q}_{l,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha-\beta_{k'}}{\beta_k-\beta_{k'}} \tag{61}$$

where $w_l(\alpha_i) = \widetilde{\mathbf{W}}_{l,i}(0)$ is the encoded model at user i, and $w_l(\beta_k) = \mathbf{W}_l(0)$ for $k\in[K]$ corresponds to the true model initialized at layer l. Model initialization can be carried out fully offline, as the initialization is random and independent from the local datasets of the users.

D. Gradient Computation

Using the encoded dataset and labels, users then compute the gradients. Let $\widetilde{\mathbf{Z}}_{l,i}(t) \in \mathbb{F}_p^{d_l\times\frac{Nm}{K}}$ and $\widetilde{\mathbf{U}}_{l,i}(t) \in \mathbb{F}_p^{d_l\times\frac{Nm}{K}}$ denote the input and output of the activation function $g(\cdot)$ at layer $l\in[L+1]$, where $\widetilde{\mathbf{U}}_{l,i}(t) = g(\widetilde{\mathbf{Z}}_{l,i}(t))$. Then, gradient computation consists of the following forward and backward propagation steps.

(Forward Propagation): For each layer $l\in[L+1]$, user $i\in[N]$ initially computes,

$$\widetilde{\mathbf{Z}}_{l,i}(t) = \widetilde{\mathbf{W}}_{l,i}(t)\widetilde{\mathbf{U}}_{l-1,i}(t) \in \mathbb{F}_p^{d_l\times\frac{Nm}{K}} \tag{62}$$

where $\widetilde{\mathbf{U}}_{0,i}(t) \triangleq \widetilde{\mathbf{X}}_i \in \mathbb{F}_p^{d\times\frac{Nm}{K}}$. As the degree of the resulting polynomial in (62) is greater than $K+T-1$, users carry out a degree reduction operation using DLC from (19),

$$\widetilde{\mathbf{Z}}_{l,1}(t),\dots,\widetilde{\mathbf{Z}}_{l,N}(t) \leftarrow DLC(\widetilde{\mathbf{Z}}_{l,1}(t),\dots,\widetilde{\mathbf{Z}}_{l,N}(t), M) \tag{63}$$

by letting,

$$M = \begin{cases} 2(K+T-1) & \text{if} \quad l=1 \\ 3(K+T-1) & \text{if} \quad l\geq 2 \end{cases} \tag{64}$$

and $f(\alpha_i) = \widetilde{\mathbf{Z}}_{l,i}(t)$ for all $i\in[N]$. At the end, each user $i\in[N]$ learns an updated coded matrix $\widetilde{\mathbf{Z}}_{l,i}(t) \leftarrow f'(\alpha_i) \in \mathbb{F}_p^{d_l\times\frac{Nm}{K}}$ corresponding to a Lagrange polynomial with degree $K+T-1$. Finally, user i computes the quadratic activation function,

$$\widetilde{\mathbf{U}}_{l,i}(t) = g(\widetilde{\mathbf{Z}}_{l,i}(t)) \in \mathbb{F}_p^{d_l\times\frac{Nm}{K}} \tag{65}$$

element-wise across the matrix $\widetilde{\mathbf{Z}}_{l,i}(t)$.

(Backward Propagation): After forward propagation, the final gradients are computed through backpropagation [49]. For each layer $l\in[L]$, backpropagation consists of an error propagation step, where each user locally computes,

$$\widetilde{\mathbf{E}}_{l,i}(t) = 2\widetilde{\mathbf{Z}}_{l,i}(t) \odot (\widetilde{\mathbf{W}}_{l+1,i}^{\mathrm{T}}(t) \times \widetilde{\mathbf{E}}_{l+1,i}(t)) \in \mathbb{F}_p^{d_l\times\frac{Nm}{K}} \tag{66}$$

where $\widetilde{\mathbf{E}}_{L+1,i}(t) \triangleq 2(\widetilde{\mathbf{Z}}_{L+1,i}(t) - \widetilde{\mathbf{Y}}_i) \in \mathbb{F}_p^{c\times\frac{Nm}{K}}$. After (66), users reduce the degree of the resulting polynomial from $3(K+T-1)$ back to $K+T-1$ using DLC from (19),

$$\widetilde{\mathbf{E}}_{l,1}(t),\dots,\widetilde{\mathbf{E}}_{l,N}(t) \leftarrow DLC(\widetilde{\mathbf{E}}_{l,1}(t),\dots,\widetilde{\mathbf{E}}_{l,N}(t), 3(K+T-1)) \tag{67}$$

at the end of which each user $i\in[N]$ receives an updated coded matrix $\widetilde{\mathbf{E}}_{l,i}(t)$ corresponding to an interpolation point of a Lagrange polynomial of degree $K+T-1$. Finally, user $i\in[N]$ locally computes the gradient at layer $l\in[L+1]$ as follows,

$$\widetilde{\mathbf{G}}_{l,i}(t) = \widetilde{\mathbf{E}}_{l,i}(t) \times \widetilde{\mathbf{U}}_{l-1,i}^{\mathrm{T}}(t) \in \mathbb{F}_p^{d_l\times d_{l-1}} \tag{68}$$

E. Model Update

After gradient computation, users update the model for the next training round. From (49), we observe that the coded dataset $\widetilde{\mathbf{X}}_i$ encodes K shards $\left\{\begin{bmatrix}\mathbf{X}_{1,k} & \cdots & \mathbf{X}_{N,k}\end{bmatrix}\right\}_{k\in[K]}$ from the true dataset $\mathbf{X}$, where each shard $\overline{\mathbf{X}}_k = \begin{bmatrix}\mathbf{X}_{1,k} & \cdots & \mathbf{X}_{N,k}\end{bmatrix} \in \mathbb{F}_p^{d\times\frac{Nm}{K}}$ consists of Nm/K data samples. Accordingly, the coded gradient $\widetilde{\mathbf{G}}_{l,i}(t)$ from (68) can be viewed as an interpolation point of a degree $3(K+T-1)$ polynomial $h_l(\alpha)$, where $h_l(\alpha_i) = \widetilde{\mathbf{G}}_{l,i}(t)$ is the coded gradient at user i, and $h_l(\beta_k) = \mathbf{G}_{l,k}(t)$ is the true gradient for the k^{th} shard $\overline{\mathbf{X}}_k$, which denotes the sum of the gradients across Nm/K data samples. On the other hand, the model update $\mathbf{W}_l(t+1)$ from (5) requires the aggregated gradient $\mathbf{G}_l(t) = \sum_{k\in[K]} \mathbf{G}_{l,k}(t)$ across all Nm data samples. In the following, we propose a privacy-preserving model update mechanism by aggregating the true gradients across the K shards without revealing their true content. The gradient aggregation process consists of the following offline and online phases.

(Offline) For each layer $l\in[L+1]$, user $i\in[N]$ first generates $3(K+T-1)+1$ random matrices $\mathbf{B}_{l,i,1}(t),\dots,\mathbf{B}_{l,i,3(K+T-1)+1}(t) \in \mathbb{F}_p^{\frac{d_l}{N-T}\times d_{l-1}}$ independently and uniformly at random, and sends an encoded matrix

$$\widetilde{\mathbf{B}}_{l,i,j}(t) = \sum_{k\in[3(K+T-1)+1]} \mathbf{B}_{l,i,k}(t) \prod_{k'\in[3(K+T-1)+1]\setminus\{k\}} \frac{\alpha_j-\beta_{k'}}{\beta_k-\beta_{k'}} \quad \forall l\in[L+1] \tag{69}$$

to user $j\in[N]$. Next, user $i\in[N]$ generates T random matrices $\mathbf{S}_{l,i,K+1}(t),\dots,\mathbf{S}_{l,i,K+T}(t) \in \mathbb{F}_p^{\frac{d_l}{N-T}\times d_{l-1}}$ independently

and uniformly at random, and sends an encoded matrix,

$$\overline{\mathbf{B}}_{l,i,j}(t) = \sum_{k\in[K]} \Big(\sum_{k'\in[K]} \mathbf{B}_{l,i,k'}(t) \Big) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{S}_{l,i,k}(t) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j - \beta_{k'}}{\beta_k - \beta_{k'}} \quad \forall l \in [L+1] \tag{70}$$

Upon receiving $\{\widetilde{\mathbf{B}}_{l,j,i}(t), \overline{\mathbf{B}}_{l,i,j}(t)\}_{j\in[N]}$, user $i \in [N]$ generates two higher-dimensional encoded matrices,

$$\widetilde{\mathbf{B}}_{l,i}(t) = \begin{bmatrix} \sum_{j\in[N]} \lambda_1^{j-1} \widetilde{\mathbf{B}}_{l,j,i}(t) \\ \vdots \\ \sum_{j\in[N]} \lambda_{N-T}^{j-1} \widetilde{\mathbf{B}}_{l,j,i}(t) \end{bmatrix} \in \mathbb{F}_p^{d_l \times d_{l-1}},$$

$$\overline{\mathbf{B}}_{l,i}(t) = \begin{bmatrix} \sum_{j\in[N]} \lambda_1^{j-1} \overline{\mathbf{B}}_{l,j,i}(t) \\ \vdots \\ \sum_{j\in[N]} \lambda_{N-T}^{j-1} \overline{\mathbf{B}}_{l,j,i}(t) \end{bmatrix} \in \mathbb{F}_p^{d_l \times d_{l-1}} \tag{71}$$

The first matrix $\widetilde{\mathbf{B}}_{l,i}(t)$ can be viewed as an interpolation point of a degree $3(K+T-1)$ (high-degree) polynomial,

$$r_l(\alpha) = \sum_{k\in[3(K+T-1)+1]} \mathbf{B}_{l,k}(t) \prod_{k'\in[3(K+T-1)+1]\setminus\{k\}} \frac{\alpha - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{72}$$

where

$$\mathbf{B}_{l,k}(t) \triangleq \begin{bmatrix} \sum_{j\in[N]} \lambda_1^{j-1} \mathbf{B}_{l,j,k}(t) \\ \vdots \\ \sum_{j\in[N]} \lambda_{N-T}^{j-1} \mathbf{B}_{l,j,k} \end{bmatrix} = r_l(\beta_k) \quad \forall k \in [3(K+T-1)+1] \tag{73}$$

and $\widetilde{\mathbf{B}}_{l,i}(t) = r_l(\alpha_i)$. The second matrix $\overline{\mathbf{B}}_{l,i}(t)$ can be viewed as an interpolation point of a degree $K+T-1$ (low-degree) polynomial,

$$u_l(\alpha) = \sum_{k\in[K]} \mathbf{B}_l(t) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{S}_{l,k}(t) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{74}$$

where $\overline{\mathbf{B}}_{l,i}(t) = u_l(\alpha_i)$,

$$\mathbf{B}_l(t) \triangleq \sum_{k'\in[K]} \mathbf{B}_{l,k'}(t) \tag{75}$$

and

$$\mathbf{S}_{l,k}(t) \triangleq \begin{bmatrix} \sum_{j\in[N]} \lambda_1^{j-1} \mathbf{S}_{l,j,k}(t) \\ \vdots \\ \sum_{j\in[N]} \lambda_{N-T}^{j-1} \mathbf{S}_{l,j,k}(t) \end{bmatrix} \quad \forall k \in \{K+1, \ldots, K+T\} \tag{76}$$

(Online) In the online phase, each user $i \in [N]$ initially broadcasts,

$$\widehat{\mathbf{G}}_{l,i}(t) \triangleq \widetilde{\mathbf{G}}_{l,i}(t) - \widetilde{\mathbf{B}}_{l,i}(t) \quad \forall l \in [L+1], \tag{77}$$

which can be viewed as an evaluation point of the degree M polynomial $h_l(\alpha) - r_l(\alpha)$ where,

$$\widehat{\mathbf{G}}_{l,i}(t) = h_l(\alpha_i) - r_l(\alpha_i) \quad \forall l \in [L+1], \tag{78}$$

Hence, after receiving (77) from any set of $3(K+T-1)+1$ users, each user can decode the masked gradients,

$$h_l(\beta_k) - r_l(\beta_k) = h_l(\beta_k) - \mathbf{B}_{l,k}(t) \quad \forall k \in [K], l \in [L+1] \tag{79}$$

Finally, user $i \in [N]$ aggregates and re-encodes the masked gradients by forming a degree $K+T-1$ (lower-degree) Lagrange interpolation polynomial as follows,

$$\widetilde{\mathbf{G}}_{l,i}(t) = \sum_{k\in[K]} \Big(\sum_{k'\in[K]} (h_l(\beta_{k'}) - \mathbf{B}_{l,k'}(t)) \Big) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \overline{\mathbf{B}}_{l,i}(t) \tag{80}$$

$$= \sum_{k\in[K]} \Big(\sum_{k'\in[K]} h_l(\beta_{k'}) \Big) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{S}_{l,k}(t) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{81}$$

$$= \sum_{k\in[K]} \mathbf{G}_l(t) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=K+1}^{K+T} \mathbf{S}_{l,k}(t) \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{82}$$

The encoded gradient $\widetilde{\mathbf{G}}_{l,i}(t)$ from (82) encodes the aggregate $\mathbf{G}_l(t) = \sum_{k\in[K]} \mathbf{G}_{l,k}(t)$ of the true gradients $\mathbf{G}_{l,1}(t), \ldots, \mathbf{G}_{l,K}(t)$ evaluated across the K shards $\{\overline{\mathbf{X}}_k\}_{k\in[K]}$ of the dataset $\mathbf{X}$.

After aggregating the gradients, the model is updated for the next training round. For the model update rule from (5), this corresponds to the following computation,

$$\widetilde{\mathbf{W}}_{l,i}(t+1) = \widetilde{\mathbf{W}}_{l,i}(t) - \frac{\eta}{Nm} \widetilde{\mathbf{G}}_{l,i}(t) \quad \forall l \in [L+1]. \tag{83}$$

Note that $\eta \ll 1$ in (83), whereas our framework is bound to finite field polynomial operations, consisting of finite field addition and multiplications only. To handle this, one approach is to consider a sufficiently large field size and convert (83) to an integer domain operation. In our theoretical analysis, we assume a sufficiently large field size and follow this approach for tractability, while providing the details in App. E, In practice, one can also leverage the secure truncation protocol from [11], [50], to reduce the required field size while handling the model update in (83), albeit with a slight loss in model accuracy due to quantization. In our experiments, we utilize the latter, and provide the implementation details in Section VIII.

(Final Model Recovery) At the end of J training rounds, parties can decode the final model $\{\mathbf{W}_l(J)\}_{l\in[L+1]}$ by collecting the coded models $\{\widetilde{\mathbf{W}}_{l,i}(J)\}_{l\in[L+1]}$ from any set of $K+T$ users, and using polynomial interpolation.

The algorithm steps of CLOVER are provided in App. 1. In the following, we present a motivating example to demonstrate the training process.

VI. Motivating Example

In this section, we present an illustrative example for CLOVER. We consider a scenario where the total number of users is $N = 11$, parallelization degree is $K = 2$, number of adversaries is $T = 2$, and number of dropout users is $D = 1$. For simplicity, we consider $m = 2$ data samples per user, $d = 10$ features per sample, and $c = 4$ classes. The neural network has $L = 1$ hidden layer, with $d_1 = 5$ neurons at layer $l = 1$. As shown in (51), each user $i \in [11]$ initially holds an encoded dataset $\widetilde{\mathbf{X}}_i \in \mathbb{F}_p^{10\times 11}$,

$$\widetilde{\mathbf{X}}_i = \sum_{k\in[2]} \overline{\mathbf{X}}_k \prod_{k'\in[4]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=3}^{4} \mathbf{V}_k \prod_{k'\in[4]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} \in \mathbb{F}_p^{d\times \frac{Nm}{2}} \quad (84)$$

which corresponds to an interpolation point of a degree $K + T - 1 = 3$ polynomial $x(\alpha)$ such that $\widetilde{\mathbf{X}}_i = x(\alpha_i)$, whereas $x(\beta_k) = \overline{\mathbf{X}}_k \in \mathbb{F}_p^{10\times 11}$ denotes the k^{th} shard of the true dataset $\mathbf{X} = [\overline{\mathbf{X}}_1 \quad \overline{\mathbf{X}}_2]$. Similarly, each user $i \in [11]$ holds the encoded labels $\widetilde{\mathbf{Y}}_i \in \mathbb{F}_p^{4\times 11}$,

$$\widetilde{\mathbf{Y}}_i = \sum_{k\in[2]} \overline{\mathbf{Y}}_k \prod_{k'\in[4]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=3}^{4} \mathbf{N}_k \prod_{k'\in[4]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} \quad (85)$$

which corresponds to an interpolation point of a degree $K + T - 1 = 3$ polynomial $y(\alpha)$ such that $\widetilde{\mathbf{Y}}_i = y(\alpha_i)$, whereas $y(\beta_k) = \overline{\mathbf{Y}}_k \in \mathbb{F}_p^{4\times 11}$ denotes the k^{th} shard of the true labels $\mathbf{Y} = [\overline{\mathbf{Y}}_1 \quad \overline{\mathbf{Y}}_2]$ as shown in (55). In addition to the encoded dataset and labels, each user $i \in [11]$ holds the encoded model $\widetilde{\mathbf{W}}_{1,i}(0) \in \mathbb{F}_p^{5\times 10}$ for the first layer, and $\widetilde{\mathbf{W}}_{2,i}(0) \in \mathbb{F}_p^{4\times 5}$ for the second layer, where

$$\widetilde{\mathbf{W}}_{l,i}(0) = \sum_{k\in[2]} \mathbf{W}_l(0) \prod_{k'\in[4]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} + \sum_{k=3}^{4} \mathbf{Q}_{l,k} \prod_{k'\in[4]\setminus\{k\}} \frac{\alpha_i-\beta_{k'}}{\beta_k-\beta_{k'}} \quad \forall l \in [2], \quad (86)$$

which corresponds to an interpolation point of a degree $K + T - 1 = 3$ polynomial $w_l(\alpha)$ such that $\widetilde{\mathbf{W}}_{l,i}(0) = w_l(\alpha_i)$ for layer $l \in [2]$, whereas $w_1(\beta_k) = \mathbf{W}_1(0) \in \mathbb{F}_p^{5\times 10}$ and $w_2(\beta_k) = \mathbf{W}_2(0) \in \mathbb{F}_p^{4\times 5}$ denote the randomly initialized secret model parameters as shown in (61). Using the encoded dataset $\widetilde{\mathbf{X}}_i$, labels $\widetilde{\mathbf{Y}}_i$, and initial model $\widetilde{\mathbf{W}}_{1,i}(0), \widetilde{\mathbf{W}}_{2,i}(0)$, users $i \in [11]$ then perform training, which consists of gradient computations and model update. In the following, we demonstrate the gradient computations for the first round $t = 0$, which consists of the following forward propagation and backpropagation steps, respectively.

(Forward propagation) For forward propagation, each user $i \in [11]$ initially computes $\widetilde{\mathbf{Z}}_{1,i}(0) = \widetilde{\mathbf{W}}_{1,i}(0)\widetilde{\mathbf{X}}_i \in \mathbb{F}_p^{5\times 11}$ for the first layer $l = 1$ as shown in (62). The local computation $\widetilde{\mathbf{Z}}_{1,i}(0)$ can be viewed as an interpolation point of the degree $2(K + T - 1) = 6$ polynomial $f(\alpha) = w_1(\alpha)x(\alpha)$, where $w_1(\alpha_i)x(\alpha_i) = \widetilde{\mathbf{Z}}_{1,i}(0)$, and $w_1(\beta_k)x(\beta_k) = \mathbf{W}_1(0)\overline{\mathbf{X}}_k \in \mathbb{F}_p^{5\times 11}$ for $k \in [2]$ refers to the secret computation corresponding to the k^{th} shard $\overline{\mathbf{X}}_k$ of the true dataset $\mathbf{X}$.

If users continue forward propagation using the encoded computations $w_1(\alpha_i)x(\alpha_i)$, the polynomial degree will double after the activation function $g(\cdot)$ from (65), and the local computation $\widetilde{\mathbf{U}}_{1,i}(0) = g(\widetilde{\mathbf{Z}}_{1,i}(0))$ will correspond to an interpolation point of a degree $2(2(K+T-1)) = 12$ polynomial $g(w_1(\alpha)x(\alpha))$. As interpolating a polynomial of degree 12 requires at least $12 + 1 = 13$ interpolation points, the total number of users $N = 11 < 13$ will not be sufficient to decode the final model. To avoid this, the degree of the polynomial $f(\alpha) = w_1(\alpha)x(\alpha)$ is reduced from $2(K + T - 1) = 6$ back to $K + T - 1 = 3$ by using DLC from (19),

$$\widetilde{\mathbf{Z}}_{1,1}(0), \ldots, \widetilde{\mathbf{Z}}_{1,11}(0) = DLC(\widetilde{\mathbf{Z}}_{1,1}(0), \ldots, \widetilde{\mathbf{Z}}_{1,11}(0), 6) \quad (87)$$

at the end of which user $i \in [11]$ receives an encoded computation $\widetilde{\mathbf{Z}}_{1,i}(0)$, which corresponds to an evaluation of a degree $K + T - 1 = 3$ polynomial $f'(\alpha)$ where $f'(\alpha_i) = \widetilde{\mathbf{Z}}_{1,i}(0)$, and

$$f'(\beta_k) = w_1(\beta_1)x(\beta_k) = \mathbf{W}_1(0)\overline{\mathbf{X}}_k \quad k \in [2] \quad (88)$$

denotes the secret computations corresponding to the $K = 2$ shards $\overline{\mathbf{X}}_1, \overline{\mathbf{X}}_2$ of the true dataset $\mathbf{X}$. After degree reduction, each user $i \in [11]$ locally computes the activation function $\widetilde{\mathbf{U}}_{1,i}(0) = g(\widetilde{\mathbf{Z}}_{1,i}(0)) \in \mathbb{F}_p^{5\times 11}$, which increases the polynomial degree to $2(K + T - 1) = 6$.

For the second layer $l = 2$, each user $i \in [11]$ computes $\widetilde{\mathbf{Z}}_{2,i}(0) = \widetilde{\mathbf{W}}_{2,i}(0)\widetilde{\mathbf{U}}_{1,i}(0)$ from (62), using the encoded output $\widetilde{\mathbf{U}}_{1,i}(0)$ of the first layer, and the encoded model $\widetilde{\mathbf{W}}_{2,i}(0)$ for the second layer. Due to the multiplication operation, each local computation $\widetilde{\mathbf{Z}}_{2,i}(0)$ is now an evaluation of a degree $3(K + T - 1) = 9$ polynomial $w_2(\alpha)g(f'(\alpha))$ such that $\widetilde{\mathbf{Z}}_{2,i}(0) = w_2(\alpha_i)g(f'(\alpha_i))$, whereas the secret computations for the two shards $\overline{\mathbf{X}}_1$, and $\overline{\mathbf{X}}_2$ are given by,

$$w_2(\beta_k)g(f'(\beta_k)) = w_2(\beta_k)g(w_1(\beta_k)x(\beta_k)) = \mathbf{W}_2(0)g(\mathbf{W}_1(0)\overline{\mathbf{X}}_k) \quad k \in [2] \quad (89)$$

To reduce the degree of the polynomial $w_2(\alpha)g(f'(\alpha))$ from $3(K+T-1) = 9$ back to $K+T-1 = 3$, users again leverage DLC from (19),

$$\widetilde{\mathbf{Z}}_{2,1}(0), \ldots, \widetilde{\mathbf{Z}}_{2,11}(0) = DLC(\widetilde{\mathbf{Z}}_{2,1}(0), \ldots, \widetilde{\mathbf{Z}}_{2,11}(0), 9) \quad (90)$$

at the end of which each user $i \in [11]$ receives an encoded computation $\widetilde{\mathbf{Z}}_{2,i}(0)$ that can be viewed as an interpolation point of a degree $K+T-1 = 3$ polynomial $f''(\alpha)$, such that $f''(\alpha_i) = \widetilde{\mathbf{Z}}_{2,i}(0)$, and true computations for the two shards $\overline{\mathbf{X}}_1$, $\overline{\mathbf{X}}_2$ of the true dataset $\mathbf{X}$ are,

$$f''(\beta_k) = w_2(\beta_k)g(f'(\beta_k)) = w_2(\beta_k)g(w_1(\beta_k)x(\beta_k)) = \mathbf{W}_2(0)g(\mathbf{W}_1(0)\overline{\mathbf{X}}_k) \quad k \in [2] \quad (91)$$

which concludes the forward propagation step. Fig. 8 illustrates the forward propagation steps for user 1. After forward propagation, users carry out the backpropagation operations as follows.

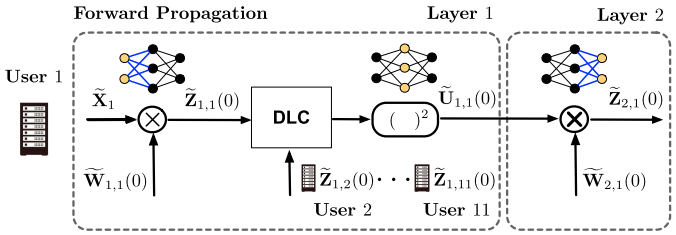


Fig. 8. Motivating example with $N = 11$ users for training a neural network with $L = 1$ hidden layer, and illustration of the forward propagation steps for user 1.

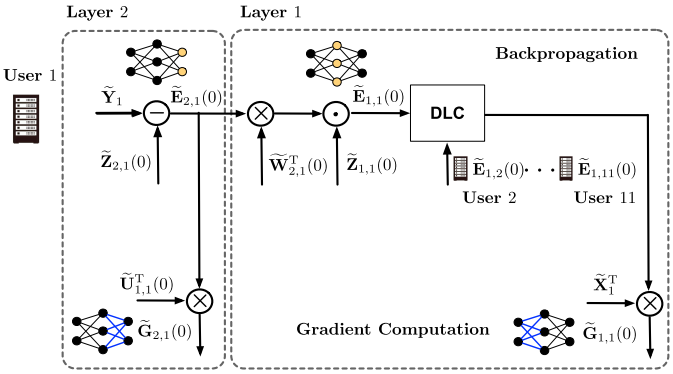


Fig. 9. Illustration of the backpropagation and gradient computation steps for user 1.

(Backpropagation) Each user $i \in [11]$ initially computes the error for the last layer $l = 2$,

$$\widetilde{\mathbf{E}}_{2,i}(0) = 2(\widetilde{\mathbf{Z}}_{2,i}(0) - \widetilde{\mathbf{Y}}_i) \tag{92}$$

The local computation $\widetilde{\mathbf{E}}_{2,i}(0)$ corresponds to an evaluation point of a degree $K+T-1=3$ polynomial $2(f''(\alpha)-y(\alpha))$ at $\alpha = \alpha_i$, such that $\widetilde{\mathbf{E}}_{2,i}(0) = 2(f''(\alpha_i) - y(\alpha_i))$, whereas the secret computations for the two shards $\overline{\mathbf{X}}_1$, $\overline{\mathbf{Y}}_1$ and $\overline{\mathbf{X}}_2$, $\overline{\mathbf{Y}}_2$ of the true dataset $\mathbf{X}$ and labels $\mathbf{Y}$ are,

$$2\big(f''(\beta_k)-y(\beta_k)\big)=2\Big(\mathbf{W}_2(0)g\big(\mathbf{W}_1(0)\overline{\mathbf{X}}_k\big)-\overline{\mathbf{Y}}_k\Big) \quad k\in[2] \tag{93}$$

where $f''(\beta_k)$ is as given in (91). Since the polynomial degree is still $K+T-1=3$, no degree reduction is required for this layer. Next, for the first layer $l = 1$, user $i \in [11]$ computes,

$$\widetilde{\mathbf{E}}_{1,i}(0) = 2\widetilde{\mathbf{Z}}_{1,i}(0) \odot (\widetilde{\mathbf{W}}_{2,i}^{\mathrm{T}}(0) \times \widetilde{\mathbf{E}}_{2,i}(0)) \tag{94}$$

The local computation $\widetilde{\mathbf{E}}_{1,i}(0)$ corresponds to an interpolation point of a degree $3(K+T-1)=9$ polynomial $2f'(\alpha) \odot \Big(w_2^{\mathrm{T}}(\alpha) \times 2\big(f''(\alpha)-y(\alpha)\big)\Big)$ at $\alpha = \alpha_i$,

$$\widetilde{\mathbf{E}}_{1,i}(0) = 2f'(\alpha_i) \odot \Big(w_2^{\mathrm{T}}(\alpha_i) \times 2\big(f''(\alpha_i)-y(\alpha_i)\big)\Big), \tag{95}$$

whereas the secret computations corresponding to the two shards $\overline{\mathbf{X}}_1$, $\overline{\mathbf{Y}}_1$ and $\overline{\mathbf{X}}_2$, $\overline{\mathbf{Y}}_2$ are,

$$\begin{aligned}&2f'(\beta_k) \odot (w_2^{\mathrm{T}}(\beta_k) \times 2(f''(\beta_k) - y(\beta_k)))\\ &= 2(\mathbf{W}_1(0)\overline{\mathbf{X}}_k) \odot \Big(\mathbf{W}_2^{\mathrm{T}}(0) \times 2\big(\mathbf{W}_2(0)g(\mathbf{W}_1(0)\overline{\mathbf{X}}_k) - \overline{\mathbf{Y}}_k\big)\Big)\end{aligned} \tag{96}$$

for $k \in [2]$, where $f'(\beta_k)$ and $f''(\beta_k)$ are given in (88) and (91), respectively. To reduce the polynomial degree back to $K+T-1=3$, users again leverage DLC from (19),

$$\widetilde{\mathbf{E}}_{1,1}(0), \ldots, \widetilde{\mathbf{E}}_{1,11}(0) = DLC(\widetilde{\mathbf{E}}_{1,1}(0), \ldots, \widetilde{\mathbf{E}}_{1,11}(0), 9) \tag{97}$$

at the end of which user $i \in [11]$ receives an encoded computation $\widetilde{\mathbf{E}}_{1,i}(0)$, which can be viewed as an interpolation point of a degree $K+T-1=3$ polynomial $f'''(\alpha)$, such that $\widetilde{\mathbf{E}}_{1,i}(0) = f'''(\alpha_i)$, whereas the secret computations for the two shards $\overline{\mathbf{X}}_1$, $\overline{\mathbf{Y}}_1$ and $\overline{\mathbf{X}}_2$, $\overline{\mathbf{Y}}_2$ are,

$$\begin{aligned}f'''(\beta_k) = 2(\mathbf{W}_1(0)\overline{\mathbf{X}}_k) \odot \Big(&\mathbf{W}_2^{\mathrm{T}}(0)\\ &\times 2\big(\mathbf{W}_2(0)g(\mathbf{W}_1(0)\overline{\mathbf{X}}_k) - \overline{\mathbf{Y}}_k\big)\Big) \quad k\in[2]\end{aligned} \tag{98}$$

from (96). Finally, user $i \in [11]$ locally computes the gradient for the first layer $l = 1$,

$$\widetilde{\mathbf{G}}_{1,i}(0) = \widetilde{\mathbf{E}}_{1,i}(0)\widetilde{\mathbf{U}}_{0,i}^{\mathrm{T}}(0) = \widetilde{\mathbf{E}}_{1,i}(0)\widetilde{\mathbf{X}}_i^{\mathrm{T}} \tag{99}$$

and for the second layer $l = 2$,

$$\widetilde{\mathbf{G}}_{2,i}(0) = \widetilde{\mathbf{E}}_{2,i}(0)\widetilde{\mathbf{U}}_{1,i}^{\mathrm{T}}(0) \tag{100}$$

The encoded gradient $\widetilde{\mathbf{G}}_{1,i}(0) \in \mathbb{F}_p^{5\times 10}$ for layer $l = 1$ from (99) corresponds to an interpolation point of a degree $3(K+T-1) = 9$ polynomial $h_1(\alpha)$ such that $\widetilde{\mathbf{G}}_{1,i}(0) = h_1(\alpha_i)$, and

$$\begin{aligned}\mathbf{G}_{1,k}(0) &\triangleq h_1(\beta_k)\\ &= \Big(2(\mathbf{W}_1(0)\overline{\mathbf{X}}_k) \odot \big(\mathbf{W}_2^{\mathrm{T}}(0)\\ &\quad \times 2(\mathbf{W}_2(0)g(\mathbf{W}_1(0)\overline{\mathbf{X}}_k) - \overline{\mathbf{Y}}_k)\big)\Big)\overline{\mathbf{X}}_k^{\mathrm{T}} \in \mathbb{F}_p^{5\times 10}\end{aligned} \tag{101}$$

denotes the true gradient at layer $l = 1$ for the k^{th} shard $\overline{\mathbf{X}}_k$, $\overline{\mathbf{Y}}_k$ for $k \in [2]$. Similarly, the encoded gradient $\widetilde{\mathbf{G}}_{2,i}(0)$ for layer $l = 2$ corresponds to an interpolation point of a degree $3(K+T-1) = 9$ polynomial $h_2(\alpha)$ such that $\widetilde{\mathbf{G}}_{2,i}(0) = h_2(\alpha_i)$, and

$$\begin{aligned}\mathbf{G}_{2,k}(0) &= h_2(\beta_k)\\ &= 2\Big(\mathbf{W}_2(0)g\big(\mathbf{W}_1(0)\overline{\mathbf{X}}_k\big)\\ &\qquad - \overline{\mathbf{Y}}_k\Big)\Big(g\big(\mathbf{W}_1(0)\overline{\mathbf{X}}_k\big)\Big)^{\mathrm{T}} \in \mathbb{F}_p^{4\times 5}\end{aligned} \tag{102}$$

denotes the true gradient at layer $l = 2$ corresponding to the k^{th} shard $\overline{\mathbf{X}}_k$, $\overline{\mathbf{Y}}_k$ for $k \in [2]$. Fig. 9 illustrates the backpropagation and gradient computation steps for user 1.

After computing the gradients, users then update the model. As observed in (101) and (102), the true gradients evaluated for the two shards $\overline{\mathbf{X}}_1, \overline{\mathbf{Y}}_1$ and $\overline{\mathbf{X}}_2, \overline{\mathbf{Y}}_2$ are embedded at two different interpolation points β_1 and β_2, where $h_l(\beta_k) = \mathbf{G}_{l,k}(0)$ is the gradient evaluated with respect to $\overline{\mathbf{X}}_k, \overline{\mathbf{Y}}_k$ for $k \in [2]$ and $l \in [2]$. On the other hand, the model update from (5) requires the aggregated gradients $\mathbf{G}_l(0) = \sum_{k\in[2]} \mathbf{G}_{l,k}(0)$ for $l \in [2]$. This is achieved by the gradient

aggregation stage from (82), at the end of which each user $i \in [11]$ obtains an encoded gradient,

$$\widetilde{\mathbf{G}}_{l,i}(0) = \sum_{k\in[2]} (\mathbf{G}_{l,1}(0) + \mathbf{G}_{l,2}(0)) \prod_{k'\in[4]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=3}^{4} \mathbf{S}_{l,k}(0) \prod_{k'\in[4]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{103}$$

$$= \sum_{k\in[2]} \mathbf{G}_l(0) \prod_{k'\in[4]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} + \sum_{k=3}^{4} \mathbf{S}_{l,k}(0) \prod_{k'\in[4]\setminus\{k\}} \frac{\alpha_i - \beta_{k'}}{\beta_k - \beta_{k'}} \tag{104}$$

which encodes the true aggregated gradient $\mathbf{G}_l(0) = \mathbf{G}_{l,1}(0) + \mathbf{G}_{l,2}(0)$ for each layer $l \in [2]$. After aggregating the gradients, users update the model as in (83) for the next training round. After J training rounds, the final model can be decoded by collecting the encoded models from any set of up to $K + T - 1 = 3$ users and using polynomial interpolation. During training, which consists of the online phases of gradient computation and model update, up to $D = 1$ users may drop out at any round. Since the total number of surviving users is $N - D = 10 \geq 3(K + T - 1) + 1$, and the degree of the intermediate polynomials never exceeds $3(K + T - 1) = 9$, the evaluations from the surviving users is sufficient to recover the final model.

VII. Theoretical Analysis

We now present the information-theoretic privacy guarantees for CLOVER.

Theorem 1 (Information-Theoretic Privacy): In a network of N users, CLOVER provides information theoretic privacy against any collusions between up to T adversarial users:

$$I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_{\mathcal{T}} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) = 0 \tag{105}$$

where $\mathcal{M}_{\mathcal{T}}$ represents the collection of all messages received or generated by the adversaries throughout the model training.

Proof: (*Sketch*) The proof follows by decomposing the set of all messages received/generated by the adversaries into five stages $\mathcal{M}_{\mathcal{T}} = \mathcal{M}_1 \cup \mathcal{M}_2 \cup \mathcal{M}_3 \cup (\cup_{t=0}^{J-1} \mathcal{M}_{4,t}) \cup (\cup_{t=0}^{J-1} \mathcal{M}_{5,t})$, where $\mathcal{M}_1$ denotes the set of messages received/generated during dataset encoding, $\mathcal{M}_2$ is the set of messages for label encoding, and $\mathcal{M}_3$ is the set of messages for model initialization. $\mathcal{M}_{4,t}$ denotes the set of messages received/generated during gradient computation at training round t, whereas $\mathcal{M}_{5,t}$ denotes the set of messages for model update at round t. Then, one can analyze the conditional mutual information at each stage separately, conditioned on all past stages.

(Dataset and label encoding) Without loss of generality, let $\mathcal{T} = \{N-T+1, \ldots, N\}$ denote the indices of the adversarial users. For the first two stages dataset and label encoding, the result follows from the invertibility of the $T \times T$ MDS matrix,

$$\mathbf{\Gamma} \triangleq \begin{bmatrix} \rho_{N-T+1,K+1} & \cdots & \rho_{N,K+1} \\ \vdots & \ddots & \vdots \\ \rho_{N-T+1,K+T} & \cdots & \rho_{N,K+T} \end{bmatrix}$$

where $\rho_{j,k} \triangleq \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j - \beta_{k'}}{\beta_k - \beta_{k'}}$ for $j \in \{N - T + 1, \ldots, N\}$ and $k \in \{K + 1, \ldots, K + T\}$ refers to the coefficients of the Lagrange polynomial from (48) and (52), respectively, which are used to encode the dataset and labels. $\mathbf{\Gamma}$ ensures that there is a bijective mapping from each distinct realization of $\{\mathbf{V}_{i,k}\}_{k\in\{K+1,\ldots,K+T\}}$ and $\{\mathbf{N}_{i,k}\}_{k\in\{K+1,\ldots,K+T\}}$ to a distinct set of masks $\{\sum_{k=K+1}^{K+T} \mathbf{V}_{i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j - \beta_{k'}}{\beta_k - \beta_{k'}}\}_{j\in\mathcal{T}}$ and $\{\sum_{k=K+1}^{K+T} \mathbf{N}_{i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j - \beta_{k'}}{\beta_k - \beta_{k'}}\}_{j\in\mathcal{T}}$ hiding the true dataset and labels belonging to an honest user $i \in \mathcal{H}$. Since $\{\mathbf{V}_{i,k}, \mathbf{N}_{i,k}\}_{k\in\{K+1,\ldots,K+T\}}$ are generated independently and uniformly at random, the corresponding masks are also uniformly random, resulting in a uniform distribution for the dataset and labels as observed by any set of T colluding adversaries, where every realization is equally likely.

(Model initialization, gradient computing, and model update) The result for the remaining three stages, model initialization, gradient computing, and model update, follow from the invertibility of two MDS matrices. The first one is the $T \times T$ matrix $\mathbf{\Gamma}$ used during the initial Lagrange encoding of the randomness generated locally by the users, as shown in (20), (56), and (69). The second one is the $(N-T)\times(N-T)$ MDS matrix,

$$\mathbf{M} = \begin{bmatrix} 1 & \cdots & 1 \\ \lambda_1 & \cdots & \lambda_{N-T} \\ \vdots & \ddots & \vdots \\ \lambda_1^{N-T-1} & \cdots & \lambda_{N-T}^{N-T-1} \end{bmatrix}$$

where $\{\lambda_i\}_{i\in[N-T]}$ corresponds to the coefficients used while combining the Lagrange coded random matrices received from other users as in (24), (25), (57), and (71). This matrix is used to combine the low-dimensional Lagrange coded random matrices to generate high-dimensional Lagrange coded random masks, while ensuring a linear amortized communication overhead during randomness generation. The generated high-dimensional randomness is then used to mask the local computations during gradient computation and degree reduction.

During degree reduction, each user $i \in [N]$ sends two coded low-dimensional random matrices, $\widetilde{\mathbf{R}}_{i,j}$ and $\overline{\mathbf{R}}_{i,j}$ to users $j \in [N]$ as shown in (21) and (23), respectively, where $\widetilde{\mathbf{R}}_{i,j}$ is generated using the high degree (degree $3(K+T-1)$) Lagrange polynomial $\phi(\cdot)$ from (20) and $\overline{\mathbf{R}}_{i,j}$ is generated using the low degree (degree $K+T-1$) Lagrange polynomial $\psi(\cdot)$ from (22). After receiving $\{\widetilde{\mathbf{R}}_{i,j}\}_{j\in[N]}$ and $\{\overline{\mathbf{R}}_{i,j}\}_{j\in[N]}$, user $i \in [N]$ combines them to generate two high-dimensional coded matrices $\widetilde{\mathbf{R}}_i$ and $\overline{\mathbf{R}}_i$ as in (24) and (25). The generated randomness is then used to decode the local gradient computations and re-encode them using a low degree polynomial. During gradient decoding, the first matrix $\widetilde{\mathbf{R}}_i$ hides the true gradient $f(\beta_k)$ with a uniformly random additive mask $\mathbf{R}_k$

for each $k \in [K]$, at the end of which users only learn a masked gradient $f(\beta_k) - \mathbf{R}_k$ for $k \in [K]$. Then, the second mask $\overline{\mathbf{R}}_i$ ensures the cancellation of the additive masks $\{\mathbf{R}_k\}_{k\in[K]}$ from the masked gradients, while simultaneously re-encoding the gradients $\{f(\beta_k)\}_{k\in[K]}$ using a lower-degree Lagrange polynomial to continue training. The T additional random matrices $\{\mathbf{A}_k\}_{k\in\{K+1,\ldots,K+T\}}$ from (27), which are embedded in the encoded matrices $\overline{\mathbf{R}}_i$ ensure that the resulting low degree polynomial preserves the privacy of the true gradients against any set of T colluding adversaries. Then, the final result follows from the chain rule of mutual information. The details of our proof are provided in App. C. □

Remark 2: Our framework can also be extended to the setting in which multiple subgroups of adversaries collude separately, as long as the total number of adversaries in each subgroup does not exceed T, and there is no overlap or collusions across different subgroups.

Next, we present the communication and computation complexity of CLOVER.

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

Proof: (Online) The per-user online communication overhead consists of: $O\big(\frac{Ndm}{K}\big)$ for dataset encoding (Stage 1); $O\big(\frac{Ncm}{K}\big)$ for label encoding (Stage 2); $O\big(\sum_{l\in[L+1]} \frac{Nmd_l}{K}\big)$ (broadcast) for forward and backward propagation, respectively, for gradient computing per training round (Stage 4); and $O\big(\sum_{l\in[L+1]} d_l d_{l-1}\big)$ per round for model update (Stage 5). Note that during gradient computation, the only communication is due to the degree reduction operation.

(Offline) The per-user offline communication overhead consists of: $O\big(\sum_{l\in[L+1]} \frac{Nd_l d_{l-1}}{N-T}\big)$ for model initialization (Stage 3); $O\big(\sum_{l\in[L+1]} \frac{N^2 md_l}{K(N-T)}\big)$ for forward and backward propagation, respectively, for gradient computing per training round (Stage 4); and $O\big(\sum_{l\in[L+1]} \frac{Nd_l d_{l-1}}{N-T}\big)$ per round for model update (Stage 5). □

Theorem 3 (Computation Complexity): The per-user computation complexity of CLOVER is,

$$O\Bigg((d+c)\frac{Nm}{K}\log^2(K+T)\log\log(K+T) \tag{106}$$

$$+ J\Bigg(\sum_{l\in[L+1]} \frac{Nm}{K} d_l(d_{l-1}+N) + \sum_{l\in[L+1]} d_l\Big(d_{l-1} + \frac{Nm}{K}\Big)\Big(\frac{N}{N-T} + K + T\Big)\log^2(K+T)\log\log(K+T)\Bigg)\Bigg) \tag{107}$$

Proof: (Sketch) The per-user computational complexity of interpolating a degree κ polynomial, and evaluating it at κ points is $O(\kappa \log^2 \kappa \log\log\kappa)$ [51]. Then, dataset and label encoding requires evaluating a degree $K + T - 1$ polynomial at N points, which has a per-user computation cost of $O(\frac{N(d+c)m}{K}\log^2(K+T)\log\log(K+T))$. Model initialization has a computation cost of $O(\sum_{l\in[L+1]} \frac{Nd_l d_{l-1}}{N-T}\log^2(K+T)\log\log(K+T) + \sum_{l\in[L+1]} Nd_l d_{l-1})$ per user. Then, at each training round $t \in \{0,\ldots,J-1\}$, gradient computing has a cost of $O\Big(\frac{Nm}{K}\Big(\sum_{l\in[L+1]} d_l d_{l-1} + \sum_{l\in[L+1]} d_l(\frac{N}{N-T} + K + T)\log^2(K+T)\log\log(K+T) + \sum_{l\in[L+1]} Nd_l\Big)\Big)$ for forward and backward propagation. Finally, model update has a computation cost of $O\Big(\sum_{l\in[L+1]} d_l d_{l-1}\Big(\frac{N}{N-T} + K + T\Big)\log^2(K+T)\log\log(K+T) + \sum_{l\in[L+1]} d_l d_{l-1} N\Big)$ per user at each training round. The result then follows by aggregating the per-round computation cost over the total number of training rounds. The details of our proof are presented in App. D. □

The *recovery threshold* is defined as the minimum number of users required to correctly decode the final model. As the offline phases can take place in advance prior to training, in the following we consider the user dropouts that occur during training, i.e., during the online phases.

Theorem 4 (Recovery Threshold): In a network of N users where up to T users are adversarial (who may collude with one another), and up to D users may drop out in each training round, the recovery threshold of CLOVER is $N \geq D + 3(K + T - 1) + 1$.

Proof: The minimum number of users required for correct model recovery is equal to the minimum number of local computations required for polynomial interpolation, which is given by $N - D \geq 3(K + T - 1) + 1$ from Section V. □

VIII. Experiments

Experimental Setup: In our experiments, we consider multi-class classification on the MNIST [52] and CIFAR-10 [53] datasets. The MNIST dataset has 60000 data samples in total, with $d = 784$ features per sample. The CIFAR-10 dataset has 50000 data samples in total, with $d = 1024$ features per sample. The datasets are distributed uniformly across the N users. For the CIFAR-10 experiments, the original images are preprocessed using a pre-trained VGG model for feature extraction, after which 25088 features are extracted for each image, which are the input for the neural network. For the deep learning model, we consider a two-layer polynomial neural network with the input layer of dimension d, which is equal to the number of features for the training samples, one hidden layer with $d_1 = 128$ neurons, and an output layer of dimension $c = 10$, which is the total number of classes, as both datasets have 10 classes.

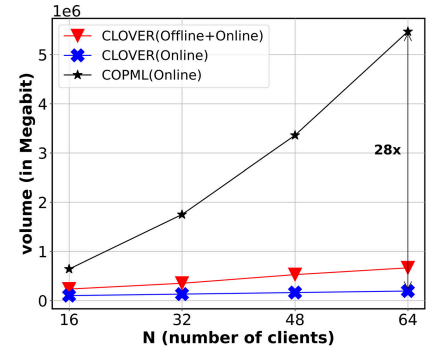
Baseline: For the baseline, we first adapt the COPML framework from [11] to neural network training as described in Section III-D, while noting that COPML was originally proposed for logistic regression. We optimize (speed-up) COPML by using the grouping strategy suggested in [11], which partitions users into groups of size $T + 1$, and communicates the secret shares only between clients within the same group to minimize the communication overhead for decoding. For

both frameworks COPML and CLOVER, we use the secure truncation operation from [50] to handle the model update in (83), specifically, the multiplication with $\frac{\eta}{m}$, to avoid an overflow in the finite field as suggested by [11]. The truncation protocol enables a secure stochastic quantization operation, where the inputs are the secret shares $\{[x]_i\}_{i\in[N]}$ of a secret x, such that client i holds a share $[x]_i$, along with two public integer parameters κ_1 and κ_2 where $0 < \kappa_1 < \kappa_2$, and $x \in \mathbb{F}_{2^{\kappa_2}}$. Then, the protocol returns the secret shares $\{[s]_i\}_{i\in[N]}$ of a variable s such that $s = \lceil \frac{x}{2^{\kappa_1}} \rceil + b$ where b is a Bernoulli random variable with probability $P[b = 1] = (x \mod 2^{\kappa_1})/2^{\kappa_1}$. As a result, the secret x is quantized by rounding $x/(2^{\kappa_1})$ to the nearest integer with probability $1-\sigma$, where σ is the distance between the two. The quantization is unbiased, which ensures the convergence of the trained model. For all datasets we use $(\kappa_1, \kappa_2) = (21, 24)$. The learning rate η is also subsumed under the truncation operation.

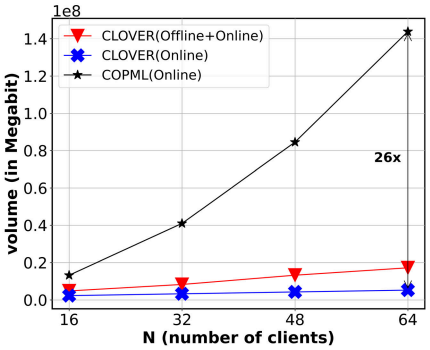
Hyperparameters: The size of the finite field is $p = 2^{26} - 5$. For the local datasets, the data samples from each class is distributed uniformly across the N users. Each user then normalizes the real-valued dataset by using the empirical mean and variance of the local data samples. Specifically, user $i \in [N]$ normalizes each feature x as $x \leftarrow (x - m_X)/\sigma_X$, where m_X and σ_X are the empirical mean and standard deviation evaluated across the data samples in the local dataset $\mathbf{X}_i$. Then, user i converts each sample and label to the finite field $\mathbb{F}_p$ as described in App. A, with the quantization parameter $\gamma = 8$. For both datasets, a randomly sampled batch of $256/K$ coded data samples are processed at each round, where sampling is uniformly at random with replacement, corresponding to 256 true data samples in total, as each coded sample encodes K true samples. For CIFAR-10, the VGG feature extractor is publicly available and pre-trained on the ImageNet dataset [54]. Each CIFAR-10 sample is first passed through the feature extractor locally by the user, which outputs 25088 features for each data sample. The resulting features are then utilized as the input to the polynomial neural network. The bandwidth is $40Mbps$.

Performance Evaluation: From Theorem 4, for correct recovery of the model, the total number of clients need to satisfy the recovery threshold $N - D \geq 3(K + T - 1) + 1$. Note that as long as the recovery threshold is satisfied, our framework ensures the correctness of the decoded gradients at each training round even if up to D users drop out from the system. In particular, after dataset encoding, each user processes a *coded version* of each data point in the dataset, hence, as long as there are $N - D$ surviving users, the local evaluations of the dropout users do not effect the accuracy of the final model. This is unlike conventional uncoded distributed learning schemes such as federated learning, where dropout users can significantly deteriorate model accuracy. As such, in the following we consider the worst-case scenario for communication overhead and privacy, where $D = 0$ and all messages are communicated across the clients. We then let $N = 3(K + T - 1) + 1$ with $K = \frac{N}{16}$ and $T = \frac{N}{8}$.

Communication Overhead: We first evaluate the total communication overhead of CLOVER across all components of training. In Fig. 10, we present the total communication volume comparing both the overall (offline+online) and online phases of CLOVER, as well as the online phases of COPML. We observe that CLOVER reduces the communication overhead by $28\times$ and $26\times$ for the MNIST and CIFAR-10 datasets, respectively.



(a) MNIST

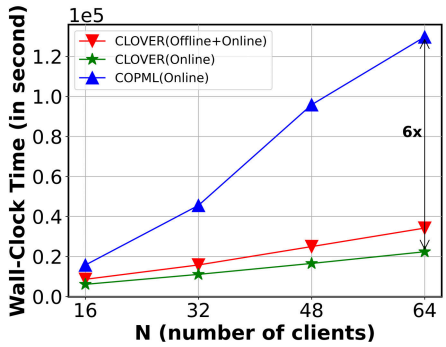


(b) CIFAR-10

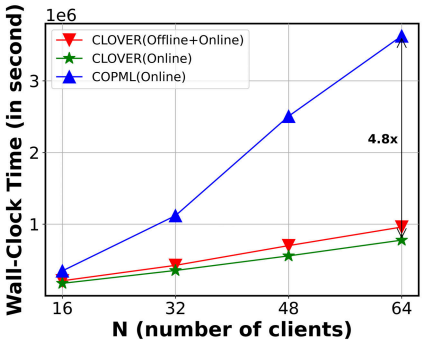
Fig. 10. Comparison of the total communication volume for the MNIST and CIFAR-10 datasets.

Wall-Clock Training Time: We next compare the wall-clock training time, including all of the communication and computation time for training, including dataset and label encoding, model initialization, gradient computation, gradient aggregation and model update. We present our results in Fig. 11, where we observe that CLOVER speeds up training by $6\times$ on the MNIST dataset, and by $4.8\times$ on CIFAR-10, respectively. As such, the wall-clock time is also greatly reduced for both datasets, even with the additional offline operations.

Model Accuracy: In Fig. 12, we demonstrate the model performance for CLOVER by measuring the test accuracy for the two datasets, with $N = 64$ clients. We then compare the model accuracy with that of a conventional feedforward neural network without privacy constraints using the ReLU activation function and cross-entropy loss with the same number of layers. We first demonstrate the performance for the conventional neural network with batch gradient descent, termed as Batch GD in Fig. 12, using the same batch of 256 true data samples that are used at each round of CLOVER, as each coded data point encodes K true data samples. This represents the performance comparison between the finite field and real domain for the batch stochastic gradient descent variant considered in our experiments. Additionally, we also demonstrate the model accuracy for the conventional neural network with full gradient

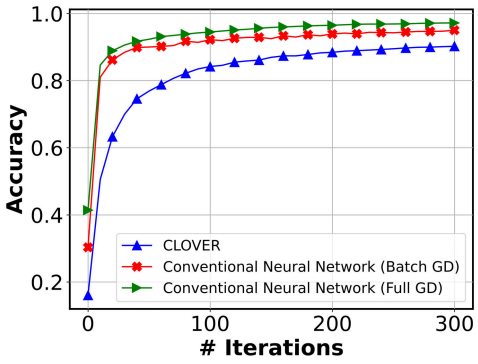


(a) MNIST

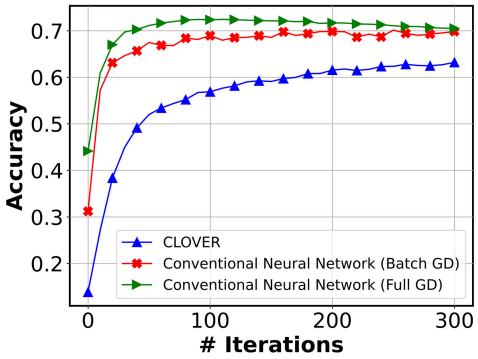


(b) CIFAR-10

Fig. 11. Comparison of the wall-clock training time for the MNIST and CIFAR-10 datasets.



(a) MNIST



(b) CIFAR-10

Fig. 12. Test accuracy of the trained model for CLOVER, for the MNIST and CIFAR-10 datasets, respectively, with respect to the baseline conventional neural network using ReLU activation and cross-entropy loss, for both batch gradient descent, termed as *Batch GD*, and full gradient descent, termed as *Full GD*.

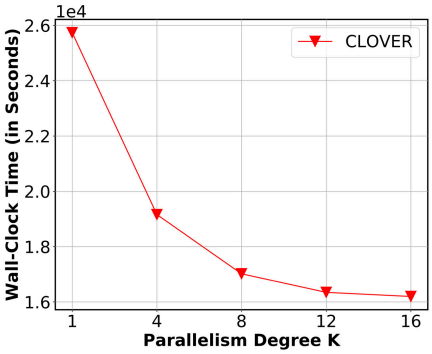
descent, termed as Full GD. We also note the accuracy reported for CLOVER in Fig. 12 is equal to the accuracy achieved by COPML, in particular, the computations carried out by the two frameworks correspond to the same real domain training updates per round.

Impact of Parallelism Degree (K): Finally, we evaluate the impact of the degree of parallelism K, by setting the number of clients to $N = 48$, and varying the parallelism degree as $K = [1, 4, 8, 12, 16]$, with the corresponding privacy level $T = [15, 11, 8, 4, 0]$, which is selected as the highest adversary tolerance level allowed by the recovery threshold $N \geq 3(K + T - 1) + 1$. In Fig. 13, we present the online wall-clock training time on the MNIST and CIFAR-10 datasets, respectively, over varying K. Our results indicate a trade-off between parallelism and adversary tolerance; increasing the degree of parallelism reduces the wall-clock training time, thus speeding up training, while decreasing the adversary tolerance.

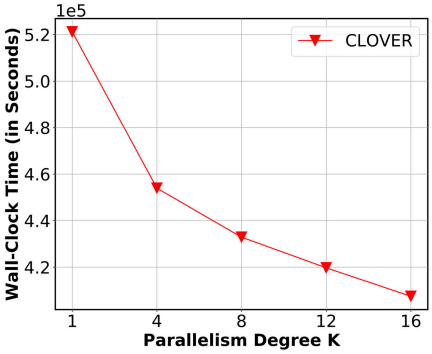
IX. Discussion

In this work, we consider feedforward polynomial neural networks with quadratic activations [45], to demonstrate the trainability of a neural network beyond the former applications limited to logistic or linear regression due to the communication complexity. Our results can further serve as a building block for more complex polynomial architectures as an interesting future direction [55], [56], [57], [58], [59], [60], which can enhance model accuracy. Another direction is to leverage conventional training architectures by approximating the nonlinear functions with polynomials. The finite field size in our experiments are constrained by the bit-width of the computational devices. Using a larger number of layers may require larger field sizes to avoid overflow errors. As such, developing software platforms to accommodate the training operations in larger fields is a promising future direction [61], [62]. Extending our mechanisms to more complex architectures may also require additional architecture and hyperparameter tuning.

Our focus in this work is on the honest-but-curious (passive) adversary model, which serves as a first step towards active and Byzantine adversaries, who can manipulate the encoding strategy as well as the messages exchanged during the protocol. A promising direction for future research is extending our frameworks for the latter, by leveraging Byzantine-resilient and verifiable secure multi-party computing mechanisms [63]. In doing so, verifiable secret sharing can be leveraged to guarantee the integrity of the encoded messages generated by the users [64]. Simultaneously, the correctness of the polynomial computations exchanged between the users can be ensured through Reed-Solomon decoding, which can identify the errors in the polynomial evaluations exchanged between the users. To facilitate correct decoding in a network with up to A active adversaries, Reed-Solomon decoding necessitates two messages per error, thereby requiring 2A additional evaluations from the remaining users for correct recovery of the trained model. Beyond the malicious modifications to



(a) MNIST



(b) CIFAR-10

Fig. 13. Wall-clock training time over varying degree of parallelism on the MNIST and CIFAR-10 datasets.

the encoding and decoding mechanism, adversaries may also target the trained model, by poisoning their local datasets to manipulate the model. Defending against such attacks further necessitates secure outlier detection mechanisms [65].

In our work, all training operations are carried out in a finite field, which leads to a trade-off between the quantization error, due to the need for converting the datasets from the real domain to the finite field, and the overflow errors during gradient computations in the finite field. One approach to extend our frameworks is to leverage privacy notions defined over the real domain, such as differential privacy. Extending our mechanisms to real-valued polynomials also requires quantifying the privacy-accuracy trade-offs, and ensuring the stability of the coded computations for training, which may require new code constructions [66].

X. Conclusion and Future Directions

This work proposes CLOVER, a collaborative privacy-preserving neural network training framework with linear communication complexity, under strong information-theoretic privacy guarantees. CLOVER builds on a novel coded computing and degree reduction mechanism, DLC, which enables efficient degree reduction for iterative applications while preserving the information-theoretic privacy of the sensitive local datasets. In doing so, CLOVER significantly improves the quadratic communication overhead of the state-of-the-art, while achieving the same adversary-resilience, robustness to user dropouts, and model accuracy.

Future directions include applications to different polynomial neural network architectures, and iterative multi-party algorithms beyond machine learning. An interesting direction is to leverage neural architecture search and hyperparameter optimization to find the best polynomial neural network to maximize the training performance. Another interesting future direction is to extend our framework to partial user dropouts, such as scenarios in which a user drops out after sending a portion of the local computations. Our mechanisms can also be extended to asynchronous learning in distributed settings, where coded samples can be processed asynchronously due to delayed or straggler users. Another direction is extending our framework to active adversaries by leveraging verifiable secret sharing and secure outlier detection mechanisms.

Appendix A
Finite Field Representation

To represent the local datasets in the finite field $\mathbb{F}_p$, user $i \in [N]$ locally quantizes each feature $x \in \mathbb{R}$ in their local dataset by employing a scalar quantization function $\Delta\left(round(2^\gamma \cdot x)\right)$ where,

$$round(x) = \begin{cases} \lfloor x \rfloor & \text{if} \quad x - \lfloor x \rfloor < 0.5 \\ \lfloor x \rfloor + 1 & \text{otherwise} \end{cases} \tag{108}$$

is a rounding operation, and γ is an integer parameter that controls the quantization loss. $\lfloor x \rfloor$ denotes the largest integer less than or equal to x, and function $\Delta : \mathbb{Z} \to \mathbb{F}_p$, represents the negative integers in the second half of the finite field,

$$\Delta(x) = \begin{cases} x & \text{if } x \geq 0 \\ p + x & \text{if } x < 0 \end{cases} \tag{109}$$

which maps the positive (negative) numbers to the first (second) half of the finite field, respectively, known as two's complement representation. Then, users represent the labels in the finite field similarly. All training computations are then performed as finite field operations in $\mathbb{F}_p$. Parameter p is chosen to be sufficiently large to avoid wrap-around in finite field computations. At the end of training, the final model $\{\mathbf{W}_l(J)\}_{l\in[L+1]}$ is mapped back from $\mathbb{F}_p$ to the real domain as $\mathbf{W}_l(J) \leftarrow \Delta^{-1}(\mathbf{W}_l(J))$. The quantization operation maps each feature $x \in \mathbb{R}$ in the local dataset to an integer value between $(-2^\gamma|x| - 1, 2^\gamma|x| + 1)$, where γ quantifies the trade-off between the quantization loss and overflow errors. A larger γ reduces the loss due to quantization, but may also increase the overflow errors. A necessary condition for the finite field size to avoid overflow errors is $p \geq 2^{(\gamma+1)} \max|x|$, where $\max|x| \in \mathbb{R}$ denotes the maximum value for any feature in the real-valued datasets.

Appendix B
Algorithms for CLOVER

The individual steps of CLOVER are presented in Algorithm 1.

Appendix C
Information-Theoretic Privacy

Proof: Consider an arbitrary set of adversaries $\mathcal{T} \subseteq N$. For ease of exposition, we focus on the worst case scenario

Algorithm 1 CLOVER

Input: Number of users N, number of features d, number of data samples m per user, number of classes c, local dataset $\mathbf{X}_i \in \mathbb{F}_p^{d\times m}$ and labels $\mathbf{Y}_i \in \mathbb{F}_p^{c\times m}$ for user $i \in [N]$, degree of parallelism K, privacy T, distinct public parameters $\alpha_1, \ldots, \alpha_N, \beta_1, \ldots \beta_K, \lambda_1, \ldots, \lambda_{N-T} \in \mathbb{F}_p$, number of hidden layers L, number of neurons d_l for layer $l \in [L+1]$.

Output: Final model $\mathbf{W}_1(J), \ldots, \mathbf{W}_{L+1}(J)$ after J training rounds.

1) Dataset Encoding

for user $i = 1, \ldots, N$
  Partition the dataset $\mathbf{X}_i \in \mathbb{F}_p^{d\times m}$ into K equal-sized shards $\mathbf{X}_{i,1}, \ldots, \mathbf{X}_{i,K} \in \mathbb{F}_p^{d\times \frac{m}{K}}$.
  for user $j = 1, \ldots, N$
    Send the encoded matrix $\widetilde{\mathbf{X}}_{i,j} \in \mathbb{F}_p^{d\times \frac{m}{K}}$ from (48) to user j.
for user $i = 1, \ldots, N$
  Generate the encoded dataset $\widetilde{\mathbf{X}}_i = \begin{bmatrix}\widetilde{\mathbf{X}}_{1,i} & \cdots & \widetilde{\mathbf{X}}_{N,i}\end{bmatrix} \in \mathbb{F}_p^{d\times \frac{Nm}{K}}$ from (49) by concatenating $\{\mathbf{X}_{j,i}\}_{j\in[N]}$.

2) Label Encoding

for user $i = 1, \ldots, N$
  Partition the labels $\mathbf{Y}_i \in \mathbb{F}_p^{c\times m}$ into K equal-sized shards $\mathbf{Y}_{i,1}, \ldots, \mathbf{Y}_{i,K} \in \mathbb{F}_p^{c\times \frac{m}{K}}$.
  for user $j = 1, \ldots, N$
    Send the encoded matrix $\widetilde{\mathbf{Y}}_{i,j} \in \mathbb{F}_p^{c\times \frac{m}{K}}$ from (52) to user j.
for user $i = 1, \ldots, N$
  Generate the encoded labels $\widetilde{\mathbf{Y}}_i = \begin{bmatrix}\widetilde{\mathbf{Y}}_{1,i} & \cdots & \widetilde{\mathbf{Y}}_{N,i}\end{bmatrix} \in \mathbb{F}_p^{c\times \frac{Nm}{K}}$ from (53) by concatenating $\{\mathbf{Y}_{j,i}\}_{j\in[N]}$.

3) Model Initialization (offline)

for user $i = 1, \ldots, N$
  for layer $l = 1, \ldots, L+1$
    Generate $T+1$ matrices $\mathbf{W}_{l,i}(0), \mathbf{Q}_l, i, K+1, \ldots, \mathbf{Q}_l, i, K+1 \in \mathbb{F}_p^{\frac{d_l}{N-T}\times d_{l-1}}$ uniformly at random.
    for user $j = 1, \ldots, N$
      Send the encoded matrix $\widetilde{\mathbf{W}}_{l,i,j}(0) \in \mathbb{F}_p^{\frac{d_l}{N-T}\times d_{l-1}}$ from (52) to user j.
for user $i = 1, \ldots, N$
  for layer $l = 1, \ldots, L+1$
    Construct the encoded model $\widetilde{\mathbf{W}}_{l,i}(0) \in \mathbb{F}_p^{d_l\times d_{l-1}}$ from (57), using $\{\widetilde{\mathbf{W}}_{l,j,i}(0)\}_{j\in[N]}$.
for iteration $t = 0, \ldots, J-1$

4) Gradient Computation

  for user $i = 1, \ldots, N$
    Set $\widetilde{\mathbf{U}}_{0,i}(t) \triangleq \widetilde{\mathbf{X}}_i \in \mathbb{F}_p^{d\times \frac{Nm}{K}}$.
  for layer $l = 1, \ldots, L+1$ //Forward Propagation
    for user $i = 1, \ldots, N$
      Compute $\widetilde{\mathbf{Z}}_{l,i}(t) = \widetilde{\mathbf{W}}_{l,i}(t)\widetilde{\mathbf{U}}_{l-1,i}(t) \in \mathbb{F}_p^{d_l\times \frac{Nm}{K}}$ from (62).
    Using DLC, reduce the degree from M to $K+T-1$ as shown in (63),

$$\widetilde{\mathbf{Z}}_{l,1}(t), \ldots, \widetilde{\mathbf{Z}}_{l,N}(t) \leftarrow DLC(\widetilde{\mathbf{Z}}_{l,1}(t), \ldots, \widetilde{\mathbf{Z}}_{l,N}(t), M) \quad \text{where } M \text{ is as defined in (64)}$$

    for user $i = 1, \ldots, N$
      Set $\widetilde{\mathbf{U}}_{l,i}(t) = g(\widetilde{\mathbf{Z}}_{l,i}(t)) \in \mathbb{F}_p^{d_l\times \frac{Nm}{K}}$ from (65).
  for user $i = 1, \ldots, N$
    Set $\widetilde{\mathbf{E}}_{L+1,i}(t) \triangleq 2(\widetilde{\mathbf{Z}}_{L+1,i}(t) - \widetilde{\mathbf{Y}}_i) \in \mathbb{F}_p^{c\times \frac{Nm}{K}}$.
  for layer $l = L, \ldots, 1$

by setting $|\mathcal{T}| = T$, while noting that the same analysis holds for all $|\mathcal{T}| < T$. Let $\mathcal{M}_1$ and $\mathcal{M}_2$, denote the collection of all messages received/generated by the adversaries during the dataset encoding (Stage 1), and label encoding (Stage 2) stages, respectively. Similarly, let $\mathcal{M}_3$ denote the collection of all messages received/generated by the adversaries during the model initialization stage (Stage 3). Finally, let $\mathcal{M}_4(t)$ denote the collection of all messages received/generated by the adversaries in the gradient computation stage (Stage 4) at training round $t \in \{0, \ldots, J-1\}$, and $\mathcal{M}_5(t)$ denote the collection of all messages received/generated by the adversaries in the model update stage (Stage 5) at training round $t \in \{0, \ldots, J-1\}$, respectively. Then, from the chain rule of mutual information, one can rewrite (105) as follows,

$$\begin{aligned}
&I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_{\mathcal{T}} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \quad (110)\\
&= I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \cup_{t=0}^{J-1}\mathcal{M}_4(t), \cup_{t=0}^{J-1}\mathcal{M}_5(t)\\
&\quad |\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \quad (111)\\
&= I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_1 | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]})\\
&+ I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_2 | \mathcal{M}_1, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]})\\
&+ I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_3 | \mathcal{M}_1, \mathcal{M}_2, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]})\\
&+ \sum_{t=0}^{J-1} I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_4(t) | \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \cup_{t'=0}^{t-1}\mathcal{M}_4(t'),\\
&\quad \cup_{t'=0}^{t-1}\mathcal{M}_5(t'), \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]})
\end{aligned}$$

for user $i = 1, \ldots, N$

Compute $\widetilde{\mathbf{E}}_{l,i}(t) = 2\widetilde{\mathbf{Z}}_{l,i}(t) \odot (\widetilde{\mathbf{W}}^{\mathrm{T}}_{l+1,i}(t) \times \widetilde{\mathbf{E}}_{l+1,i}(t)) \in \mathbb{F}_p^{d_l \times \frac{Nm}{K}}$ from (66). //Backpropagation

Using DLC, reduce the degree from $3(K+T-1)$ to $K+T-1$ as shown in (67),

$$\widetilde{\mathbf{E}}_{l,1}(t), \ldots, \widetilde{\mathbf{E}}_{l,N}(t) \leftarrow DLC(\widetilde{\mathbf{E}}_{l,1}(t), \ldots, \widetilde{\mathbf{E}}_{l,N}(t), 3(K+T-1))$$

for user $i = 1, \ldots, N$

Compute the gradient $\widetilde{\mathbf{G}}_{l,i}(t) = \widetilde{\mathbf{E}}_{l,i}(t) \times \widetilde{\mathbf{U}}^{\mathrm{T}}_{l-1,i}(t) \in \mathbb{F}_p^{d_l \times d_{l-1}}$ from (68).

5) Model Update

Offline

for layer $l = 1, \ldots, L+1$

for user $i = 1, \ldots, N$

Generate $\mathbf{B}_{l,i,1}(t), \ldots, \mathbf{B}_{l,i,3(K+T-1)+1}(t), \mathbf{S}_{l,i,K+1}(t), \ldots, \mathbf{S}_{l,i,K+T}(t) \in \mathbb{F}_p^{\frac{d_l}{N-T} \times d_{l-1}}$ uniformly random.

for user $j = 1, \ldots, N$

Send the encoded matrices $\widetilde{\mathbf{B}}_{l,i,j}(t)$ from (69) and $\overline{\mathbf{B}}_{l,i,j}(t)$ from (70) to user j.

for user $i = 1, \ldots, N$

Generate the higher-dimensional encoded matrices $\widetilde{\mathbf{B}}_{l,i}(t), \overline{\mathbf{B}}_{l,i}(t) \in \mathbb{F}_p^{d_l \times d_{l-1}}$ as in (71).

Online

for layer $l = 1, \ldots, L+1$

for user $i = 1, \ldots, N$

Broadcast the masked coded gradient $\widehat{\mathbf{G}}_{l,i}(t) = \widetilde{\mathbf{G}}_{l,i}(t) - \widetilde{\mathbf{B}}_{l,i}(t)$ from (76).

for user $i = 1, \ldots, N$

Decode the masked gradients $h_l(\beta_k) - \mathbf{B}_{l,k}(t)$ for $k \in [K]$ using polynomial interpolation.

Construct the aggregated gradient $\widetilde{\mathbf{G}}_{l,i}(t)$ using a degree $K+T-1$ Lagrange polynomial as in (80).

Using the gradient $\widetilde{\mathbf{G}}_{l,i}(t)$, update the model $\widetilde{\mathbf{W}}_{l,i}(t+1)$ as described in (83).

Final Model Recovery

Collect $\widetilde{\mathbf{W}}_{1,i}(J), \ldots, \widetilde{\mathbf{W}}_{L+1,i}(J)$ from any set of $K+T$ users, and decode the final model $\mathbf{W}_1(J), \ldots, \mathbf{W}_{L+1}(J)$.

$$+ \sum_{t=0}^{J-1} I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_5(t) | \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \cup_{t'=0}^{t} \mathcal{M}_4(t'), \cup_{t'=0}^{t-1} \mathcal{M}_5(t'), \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \quad (112)$$

We next investigate each term in the summation (112).

A. Stage 1: Dataset Encoding

We first consider the first term in (112), which corresponds to the dataset encoding stage, which can be rewritten as,

$$I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_1 | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \quad (113)$$

$$= I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \{\widetilde{\mathbf{X}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{V}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]})$$

$$= H(\{\widetilde{\mathbf{X}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{V}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]})$$

$$- H(\{\widetilde{\mathbf{X}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{V}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in[N]}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \quad (114)$$

We next bound the first term in (114) as follows:

$$H(\{\widetilde{\mathbf{X}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{V}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]})$$

$$\leq H(\{\widetilde{\mathbf{X}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{V}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}) \quad (115)$$

$$\leq \Big(\sum_{i\in\mathcal{H}} \frac{Tdm}{K} + \sum_{i\in\mathcal{T}} \frac{Tdm}{K}\Big) \log p \quad (116)$$

where (115) holds since conditioning cannot increase entropy; (116) follows from the fact that uniform distribution maximizes entropy, and that the entropy of a uniform random variable distributed over an alphabet $\mathcal{S}$ is equal to $\log|\mathcal{S}|$.

For the second term in (114), we observe that:

$$H(\{\widetilde{\mathbf{X}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{V}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}} | \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in[N]}, \{\mathbf{W}_l(J)\}_{l\in[L+1]})$$

$$= H(\{\sum_{k=K+1}^{K+T} \mathbf{V}_{i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j - \beta_{k'}}{\beta_k - \beta_{k'}}\}_{i\in\mathcal{H},j\in\mathcal{T}}) + H(\{\mathbf{V}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}) \quad (117)$$

$$= \sum_{i\in\mathcal{H}} H(\{\overline{\mathbf{V}}_{i,j}\}_{j\in\mathcal{T}}) + \frac{Td}{K}(\sum_{i\in\mathcal{T}} m) \log p \quad (118)$$

where (117) holds since given $\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in[N]}$, there is no uncertainty remaining in $\{\mathbf{X}_{ik}\}_{i\in[N],k\in[K]}$, and that the randomness is generated independently from the datasets; (118) follows from the entropy of uniform random variables, along with,

$$\overline{\mathbf{V}}_{i,j} \triangleq \sum_{k=K+1}^{K+T} \mathbf{V}_{i,k} \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j - \beta_{k'}}{\beta_k - \beta_{k'}} \quad \forall i \in \mathcal{H}, j \in \mathcal{T}. \quad (119)$$

For notational simplicity, in the following we let $\mathcal{H} = [N-T]$ and $\mathcal{T} = \{N-T+1, \ldots, N\}$, and note that same analysis holds for any set of adversarial users $\mathcal{T}$ of size T. We then represent the Lagrange polynomial coefficients as:

$$\rho_{j,k} \triangleq \prod_{k'\in[K+T]\setminus\{k\}} \frac{\alpha_j - \beta_{k'}}{\beta_k - \beta_{k'}} \quad (120)$$

for all $j \in [N]$ and $k \in [K+T]$. Then, one can write,

$$\begin{bmatrix}\overline{\mathbf{V}}_{i,N-T+1}[s] & \cdots & \overline{\mathbf{V}}_{i,N}[s]\end{bmatrix} = \begin{bmatrix}\mathbf{V}_{i,K+1}[s] & \cdots & \mathbf{V}_{i,K+T}[s]\end{bmatrix} \mathbf{\Gamma} \quad (121)$$

where $\mathbf{V}_{i,k}[s]$ and $\overline{\mathbf{V}}_{i,k}[s]$ denote the s^{th} column of $\mathbf{V}_{i,k}$ and $\overline{\mathbf{V}}_{i,k}$, respectively for $s \in [\frac{m}{K}]$, and

$$\mathbf{\Gamma} \triangleq \begin{bmatrix} \rho_{N-T+1,K+1} & \cdots & \rho_{N,K+1} \\ \vdots & \ddots & \vdots \\ \rho_{N-T+1,K+T} & \cdots & \rho_{N,K+T} \end{bmatrix} \tag{122}$$

is a $T \times T$ MDS matrix (hence is invertible), due to the MDS property of Lagrange coefficients as shown in [10]. Then,

$$H(\{\overline{\mathbf{V}}_{i,j}\}_{j\in\mathcal{T}}) = H(\{\overline{\mathbf{V}}_{i,N-T+1}[s],\ldots,\overline{\mathbf{V}}_{i,N}[s]\}_{s\in[m/K]}) \tag{123}$$

$$= H(\{\mathbf{V}_{i,K+1}[s],\ldots,\mathbf{V}_{i,K+T}[s]\}_{s\in[m/K]}) \tag{124}$$

$$= \frac{Tdm}{K}\log p \tag{125}$$

where (124) follows from (202) and that $\mathbf{\Gamma}$ is an MDS matrix, and (125) follows from the fact that each element of $\mathbf{V}_{i,k}$ is distributed uniformly at random over the finite field $\mathbb{F}_p$. By combining (125) with (118), we have that,

$$\sum_{i\in\mathcal{H}} H(\{\overline{\mathbf{V}}_{i,j}\}_{j\in\mathcal{T}}) + \frac{Td}{K}(\sum_{i\in\mathcal{T}} m)\log p = \frac{Td}{K}\Big(\sum_{i\in\mathcal{H}} m\Big)\log p + \frac{Td}{K}\Big(\sum_{i\in\mathcal{T}} m\Big)\log p \tag{126}$$

Finally, by combining (116) and (126) with (114), we find that,

$$0 \le I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\mathcal{M}_1|\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \le 0 \tag{127}$$

where the first inequality follows from the non-negativity of mutual information. Therefore, the first term in (112) satisfies:

$$I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\mathcal{M}_1|\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) = 0 \tag{128}$$

B. Stage 2: Label Encoding

We next consider the second term in (112), which corresponds to the label encoding stage,

$$\begin{aligned} &I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\mathcal{M}_2|\mathcal{M}_1,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &= I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\{\widetilde{\mathbf{Y}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}},\{\mathbf{N}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}| \\ &\qquad \mathcal{M}_1,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &= H(\{\widetilde{\mathbf{Y}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}},\{\mathbf{N}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}| \\ &\qquad \mathcal{M}_1,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &\quad - H(\{\widetilde{\mathbf{Y}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}},\{\mathbf{N}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}| \\ &\qquad \mathcal{M}_1,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \end{aligned} \tag{129}$$

The first term in (129) can be bounded as:

$$\begin{aligned} &H(\{\widetilde{\mathbf{Y}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}},\{\mathbf{N}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}| \\ &\qquad \mathcal{M}_1,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &\le H(\{\widetilde{\mathbf{Y}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}},\{\mathbf{N}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}) \end{aligned} \tag{130}$$

$$\le \Big(\sum_{i\in\mathcal{H}}\frac{Tcm}{K} + \sum_{i\in\mathcal{T}}\frac{Tcm}{K}\Big)\log p \tag{131}$$

where (130) holds since conditioning cannot increase entropy; (131) follows from the fact that uniform distribution maximizes entropy. For the second term in (129), we observe that:

$$\begin{aligned} &H(\{\widetilde{\mathbf{Y}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}},\{\mathbf{N}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}| \\ &\qquad \mathcal{M}_1,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &= H\Big(\Big\{\sum_{k=K+1}^{K+T}\mathbf{N}_{i,k}\prod_{k'\in[K+T]\setminus\{k\}}\frac{\alpha_j-\beta_{k'}}{\beta_k-\beta_{k'}}\Big\}_{i\in\mathcal{H},j\in\mathcal{T}}\Big) \\ &\qquad + H(\{\mathbf{N}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}) \end{aligned} \tag{132}$$

$$= \sum_{i\in\mathcal{H}} H(\{\overline{\mathbf{N}}_{i,j}\}_{j\in\mathcal{T}}) + \frac{Tc}{K}\Big(\sum_{i\in\mathcal{T}} m\Big)\log p \tag{133}$$

where (132) holds since given $\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]}$, there is no uncertainty remaining in $\{\mathbf{Y}_{i,k}\}_{i\in[N],k\in[K]}$, and that the randomness is generated independently from the datasets; (133) follows from the entropy of uniform random variables, and

$$\overline{\mathbf{N}}_{i,j} \triangleq \sum_{k=K+1}^{K+T}\mathbf{N}_{i,k}\prod_{k'\in[K+T]\setminus\{k\}}\frac{\alpha_j-\beta_{k'}}{\beta_k-\beta_{k'}} \qquad \forall i\in\mathcal{H}, j\in\mathcal{T}. \tag{134}$$

Then, we can write,

$$\begin{bmatrix}\overline{\mathbf{N}}_{i,N-T+1}[s] & \cdots & \overline{\mathbf{N}}_{i,N}[s]\end{bmatrix} = \begin{bmatrix}\mathbf{N}_{i,K+1}[s] & \cdots & \mathbf{N}_{i,K+T}[s]\end{bmatrix}\mathbf{\Gamma} \tag{135}$$

where $\mathbf{N}_{i,k}[s]$ and $\overline{\mathbf{N}}_{i,k}[s]$ is the s^{th} column of $\mathbf{N}_{i,k}$ and $\overline{\mathbf{N}}_{i,k}$, respectively for $s \in [\frac{m}{K}]$, and $\mathbf{\Gamma}$ is a $T\times T$ MDS matrix (hence is invertible) as defined in (122). Then,

$$H(\{\overline{\mathbf{N}}_{i,j}\}_{j\in\mathcal{T}}) = H(\{\overline{\mathbf{N}}_{i,N-T+1}[s],\ldots,\overline{\mathbf{N}}_{i,N}[s]\}_{s\in[m/K]}) \tag{136}$$

$$= H(\{\mathbf{N}_{i,K+1}[s],\ldots,\mathbf{N}_{i,K+T}[s]\}_{s\in[m/K]}) \tag{137}$$

$$= \frac{Tcm}{K}\log p \tag{138}$$

where (137) follows from (135) and that $\mathbf{\Gamma}$ is an MDS matrix, and (138) holds since each element of $\mathbf{N}_{i,k}$ is distributed uniformly at random over $\mathbb{F}_p$. By combining (138) with (133), we have,

$$\sum_{i\in\mathcal{H}} H(\{\overline{\mathbf{N}}_{i,j}\}_{j\in\mathcal{T}}) + \frac{Tc}{K}\Big(\sum_{i\in\mathcal{T}} m\Big)\log p = \frac{Tc}{K}\Big(\sum_{i\in\mathcal{H}} m\Big)\log p + \frac{Tc}{K}\Big(\sum_{i\in\mathcal{T}} m\Big)\log p \tag{139}$$

Finally, by combining (131) and (139) with (129), we have,

$$0 \le I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\mathcal{M}_2|\mathcal{M}_1,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \le 0$$

hence the second term in (112) is also equal to 0.

C. Stage 3: Model Initialization

We next consider the third term in (112), which corresponds to the model initialization stage. From (112), we find that,

$$I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\mathcal{M}_3|\mathcal{M}_1,\mathcal{M}_2,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})$$
$$= I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}},$$
$$\{\widetilde{\mathbf{W}}_{l,i,j}(0)\}_{\substack{i\in\mathcal{H},j\in\mathcal{T},\\ l\in[L+1]}},|\mathcal{M}_1,\mathcal{M}_2,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \tag{140}$$

$$= H(\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}},\{\widetilde{\mathbf{W}}_{l,i,j}(0)\}_{\substack{i\in\mathcal{H},j\in\mathcal{T}\\ l\in[L+1]}}|$$
$$\mathcal{M}_1,\mathcal{M}_2,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})$$
$$-H(\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}},\{\widetilde{\mathbf{W}}_{l,i,j}(0)\}_{\substack{i\in\mathcal{H},j\in\mathcal{T}\\ l\in[L+1]}}|$$
$$\mathcal{M}_1,\mathcal{M}_2,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \tag{141}$$

$$\leq \sum_{l\in[L+1]}\Big(Td_ld_{l-1}+\frac{Td_ld_{l-1}}{N-T}+\frac{T^2d_ld_{l-1}}{N-T}\Big)\log p$$
$$-H(\{\widetilde{\mathbf{W}}_{l,i,j}(0)\}_{\substack{i\in\mathcal{H},j\in\mathcal{T}\\ l\in[L+1]}}|\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}},$$
$$\mathcal{M}_1,\mathcal{M}_2,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]})$$
$$-H(\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}}|\mathcal{M}_1,\mathcal{M}_2,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},$$
$$\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \tag{142}$$

where the last inequality holds since conditioning cannot increase entropy, and that uniform distribution maximizes entropy. We next bound the last two terms in (142). First, observe that,

$$H(\{\widetilde{\mathbf{W}}_{l,i,j}(0)\}_{\substack{i\in\mathcal{H},j\in\mathcal{T}\\ l\in[L+1]}}|\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}},$$
$$\mathcal{M}_1,\mathcal{M}_2,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]})$$
$$\geq H(\{\widetilde{\mathbf{W}}_{l,i,j}(0)\}_{\substack{i\in\mathcal{H},j\in\mathcal{T}\\ l\in[L+1]}}|\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}},$$
$$\mathcal{M}_1,\mathcal{M}_2,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]},$$
$$\{\mathbf{W}_{l,i}(0)\}_{i\in\mathcal{H},l\in[L+1]})$$
$$= H(\{(\mathbf{Q}_{l,i,K+1}[j],\ldots,\mathbf{Q}_{l,i,K+T}[j])\mathbf{\Gamma}\}_{j\in[d_{l-1}],l\in[L+1],i\in\mathcal{H}})$$
$$= H(\{(\mathbf{Q}_{l,i,K+1}[j],\ldots,\mathbf{Q}_{l,i,K+T}[j])\}_{j\in[d_{l-1}],l\in[L+1],i\in\mathcal{H}}) \tag{143}$$

$$= \sum_{l\in[L+1]} Td_ld_{l-1}\log p \tag{144}$$

where (143) holds since $\mathbf{\Gamma}$ is an MDS matrix (hence is invertible) as shown in (122), and (144) follows from the entropy of uniform distribution. Next, for the last term in (142), we have,

$$H(\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}}|\mathcal{M}_1,\mathcal{M}_2,$$
$$\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]})$$
$$\geq H(\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}}|\mathcal{M}_1,\mathcal{M}_2,$$
$$\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]},\{\mathbf{W}_l(0)\}_{l\in[L+1]})$$
$$= H(\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}}|\{\mathbf{W}_l(0)\}_{l\in[L+1]}) \tag{145}$$

$$= H(\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}}) \tag{146}$$

$$= \sum_{l\in[L+1]}\Big(\frac{Td_ld_{l-1}}{N-T}+\frac{T^2d_ld_{l-1}}{N-T}\Big)\log p \tag{147}$$

where (145) is from the Markov chain $\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{i\in\mathcal{T},l\in[L+1],k\in\{K+1,\ldots,K+T\}} - \{\mathbf{W}_l(0)\}_{l\in[L+1]} - \mathcal{M}_1,\mathcal{M}_2,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]}$ and (147) follows from the entropy of uniform distribution. Finally, (146) follows from the fact that,

$$0\leq I(\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{\substack{i\in\mathcal{T},l\in[L+1],\\ k\in\{K+1,\ldots,K+T\}}};\mathbf{W}(0))\leq 0 \tag{148}$$

since uniform distribution maximizes entropy,

$$H(\mathbf{W}(0))\leq\sum_{l\in[L+1]}d_ld_{l-1}\log p \tag{149}$$

and

$$H(\mathbf{W}(0)|\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{i\in\mathcal{T},l\in[L+1],k\in\{K+1,\ldots,K+T\}})$$
$$= H\Big(\Big\{\sum_{i\in[N]}\lambda_1^{i-1}\mathbf{W}_{l,i}(0),\ldots,\sum_{i\in[N]}\lambda_{N-T}^{i-1}\mathbf{W}_{l,i}(0)\Big\}_{l\in[L+1]}\Big|$$
$$\{\mathbf{W}_{l,i}(0),\mathbf{Q}_{l,i,k}\}_{i\in\mathcal{T},l\in[L+1],k\in\{K+1,\ldots,K+T\}}\Big) \tag{150}$$

$$= H\Big(\Big\{\sum_{i\in[N-T]}\lambda_1^{i-1}\mathbf{W}_{l,i}(0),\ldots,$$
$$\sum_{i\in[N-T]}\lambda_{N-T}^{i-1}\mathbf{W}_{l,i}(0)\}_{l\in[L+1]}\Big|\{\mathbf{W}_{l,i}(0),$$
$$\mathbf{Q}_{l,i,k}\}_{i\in\mathcal{T},l\in[L+1],k\in\{K+1,\ldots,K+T\}}\Big) \tag{151}$$

$$= H(\{(\mathbf{W}_{l,1}[j],\ldots,\mathbf{W}_{l,N-T}[j])\mathbf{M}\}_{j\in[d_{l-1}],l\in[L+1]}) \tag{152}$$

$$= H(\{\mathbf{W}_{l,1}[j],\ldots,\mathbf{W}_{l,N-T}[j]\}_{j\in[d_{l-1}],l\in[L+1]}) \tag{153}$$

$$= \sum_{l\in[L+1]} d_ld_{l-1}\log p \tag{154}$$

where (151) follows from the independence of the randomness generated by the honest users from the adversaries. In (152), we let $\mathbf{W}_{l,i}[j]$ denote the j^{th} column of $\mathbf{W}_{l,i}(0)$, and define,

$$\mathbf{M}=\begin{bmatrix}1 & \cdots & 1\\ \lambda_1 & \cdots & \lambda_{N-T}\\ \vdots & \ddots & \vdots\\ \lambda_1^{N-T-1} & \cdots & \lambda_{N-T}^{N-T-1}\end{bmatrix} \tag{155}$$

which is an $(N-T)\times(N-T)$ MDS matrix (hence is invertible), from which (153) follows. Finally, (154) follows from the entropy of the uniform distribution. By combining (147) and (144) with (142), we have,

$$0\leq I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\mathcal{M}_3|\mathcal{M}_1,\mathcal{M}_2,$$
$$\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})$$
$$\leq 0$$

hence the third term in (112) is also equal to 0.

D. Stage 4: Gradient Computation

We next study the fourth term in (112), which corresponds to the gradient computation stage. During this stage, the only communication between the users occurs during the degree reduction phase. Therefore, without loss of generality, one can

denote the set of all messages received or generated during this stage as follows,

$$\mathcal{M}_4(t) = \mathcal{M}_{4,1}(t) \cup \cdots \cup \mathcal{M}_{4,2L+1}(t) \tag{156}$$

where $\mathcal{M}_{4,j}(t)$ denotes the messages received or generated by the adversaries during the j^{th} degree reduction operation for $j \in [2L+1]$, and $2L+1$ is the total number of degree reduction operations, as $L+1$ degree reduction operations are performed during forward pass and L operations are performed during backpropagation, respectively. Then, from (21), (23), and (29), one can rewrite the fourth term in (112) as follows,

$$\begin{aligned} &I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_4(t) | \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \cup_{t'=0}^{t-1}\mathcal{M}_4(t'), \\ &\qquad \cup_{t'=0}^{t-1}\mathcal{M}_5(t'), \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &= I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_{4,1}(t) \cup \cdots \cup \mathcal{M}_{4,2L+1}(t) | \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \\ &\cup_{t'=0}^{t-1}\mathcal{M}_4(t'), \cup_{t'=0}^{t-1}\mathcal{M}_5(t'), \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \end{aligned} \tag{157}$$

$$\begin{aligned} &= \sum_{l'\in[2L+1]} I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_{4,l'}(t) | \mathcal{M}_{4,l'-1}(t) \cup \cdots \cup \mathcal{M}_{4,1}(t), \\ &\qquad \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \cup_{t'=0}^{t-1}\mathcal{M}_4(t'), \cup_{t'=0}^{t-1}\mathcal{M}_5(t'), \\ &\qquad \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \end{aligned} \tag{158}$$

In the following, we show that the degree reduction operation with DLC from Section IV preserves information-theoretic privacy while reducing the degree of an arbitrary polynomial function $f(\cdot) \in \mathbb{F}_p^{n_1\times n_2}$ of degree M, where $f(\beta_k)$ for $k \in [K]$ represent the desired (secret) computations, and $f(\alpha_i)$ represents the local (coded) computation evaluated by user $i \in [N]$, respectively. Without loss of generality, let

$$\begin{aligned} \mathcal{M}_{l'} \triangleq (&\mathcal{M}_{4,l'-1}(t), \cdots, \mathcal{M}_{4,1}(t), \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \\ &\cup_{t'=0}^{t-1}\mathcal{M}_4(t'), \cup_{t'=0}^{t-1}\mathcal{M}_5(t')) \end{aligned} \tag{159}$$

denote the set of all messages received or generated by the adversaries prior to the degree reduction operation $l' \in [2L+1]$ at round t. Then, one can rewrite each term in (158) as follows,

$$\begin{aligned} &I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_{4,l'}(t) | \mathcal{M}_{4,l'-1}(t) \cup \cdots \cup \mathcal{M}_{4,1}(t), \mathcal{M}_1, \mathcal{M}_2, \\ &\mathcal{M}_3, \cup_{t'=0}^{t-1}\mathcal{M}_4(t'), \cup_{t'=0}^{t-1}\mathcal{M}_5(t'), \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &= I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_{4,l'}(t) | \mathcal{M}_{l'}, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \end{aligned} \tag{160}$$

$$\begin{aligned} &= I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \{\widetilde{\mathbf{R}}_{i,j}, \overline{\mathbf{R}}_{i,j}\}_{i\in\mathcal{H}, j\in\mathcal{T}}, \{f(\alpha_i) - \widetilde{\mathbf{R}}_i\}_{i\in[N]}, \\ &\qquad \{\mathbf{R}_{i,k}\}_{i\in\mathcal{T}, k\in[M+1]}, \{\mathbf{A}_{i,k}\}_{i\in\mathcal{T}, k\in\{K+1,\ldots,K+T\}} | \mathcal{M}_{l'}, \\ &\qquad \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \end{aligned} \tag{161}$$

$$\begin{aligned} &= I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \{\widetilde{\mathbf{R}}_{i,j}, \overline{\mathbf{R}}_{i,j}\}_{\substack{i\in\mathcal{H}, \\ j\in\mathcal{T}}}, \{f(\beta_k) - \mathbf{R}_k\}_{k\in[M+1]}, \\ &\qquad \{\mathbf{R}_{i,k}\}_{i\in\mathcal{T}, k\in[M+1]}, \{\mathbf{A}_{i,k}\}_{i\in\mathcal{T}, k\in\{K+1,\ldots,K+T\}} | \\ &\qquad \mathcal{M}_{l'}, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \end{aligned} \tag{162}$$

$$\begin{aligned} &= H(\{\widetilde{\mathbf{R}}_{i,j}, \overline{\mathbf{R}}_{i,j}\}_{\substack{i\in\mathcal{H}, \\ j\in\mathcal{T}}}, \{f(\beta_k) - \mathbf{R}_k\}_{k\in[M+1]}, \{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T}, \\ k\in[M+1]}}, \\ &\{\mathbf{A}_{i,k}\}_{i\in\mathcal{T}, k\in\{K+1,\ldots,K+T\}} | \mathcal{M}_{l'}, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &- H(\{\widetilde{\mathbf{R}}_{i,j}, \overline{\mathbf{R}}_{i,j}\}_{\substack{i\in\mathcal{H}, \\ j\in\mathcal{T}}}, \{f(\beta_k) - \mathbf{R}_k\}_{k\in[M+1]}, \{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T}, \\ k\in[M+1]}}, \\ &\{\mathbf{A}_{i,k}\}_{i\in\mathcal{T}, k\in\{K+1,\ldots,K+T\}} | \mathcal{M}_{l'}, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in[N]}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \end{aligned} \tag{163}$$

where (162) follows from the fact that there is a bijective mapping from any $M+1$ evaluation points $\{f(\beta_k) - \mathbf{R}_k\}_{k\in[M+1]}$ to a valid (feasible) set of local computations $\{f(\alpha_i) - \widetilde{\mathbf{R}}_i\}_{i\in[N]}$, since the local computations in (29) correspond to evaluations of a degree M monomial $\varphi(\alpha) = f(\alpha) - \phi(\alpha)$, which can be uniquely reconstructed from any set of at least $M+1$ evaluation points. For the first term in (163), we find that,

$$\begin{aligned} &H(\{\widetilde{\mathbf{R}}_{i,j}, \overline{\mathbf{R}}_{i,j}\}_{i\in\mathcal{H}, j\in\mathcal{T}}, \{f(\beta_k) - \mathbf{R}_k\}_{k\in[M+1]}, \{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T}, \\ k\in[M+1]}}, \\ &\{\mathbf{A}_{i,k}\}_{i\in\mathcal{T}, k\in\{K+1,\ldots,K+T\}} | \mathcal{M}_{l'}, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &= H(\{\widetilde{\mathbf{R}}_{i,j}, \overline{\mathbf{R}}_{i,j}\}_{i\in\mathcal{H}, j\in\mathcal{T}}, \{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T}, \\ k\in[M+1]}}, \\ &\Big\{f(\beta_k) - \Big(\sum_{i\in[N]} \lambda_1^{i-1}\mathbf{R}_{i,k}^{\mathrm{T}}, \ldots, \sum_{i\in[N]} \lambda_{N-T}^{i-1}\mathbf{R}_{i,k}^{\mathrm{T}}\Big)^{\mathrm{T}}\Big\}_{k\in[M+1]}, \\ &\{\mathbf{A}_{i,k}\}_{i\in\mathcal{T}, k\in\{K+1,\ldots,K+T\}} | \mathcal{M}_{l'}, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \end{aligned} \tag{164}$$

$$\begin{aligned} &= H(\{\overline{\mathbf{R}}_{i,j}\}_{i\in\mathcal{H}, j\in\mathcal{T}}, \{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T}, \\ k\in[M+1]}}, \Big\{f(\beta_k) - \\ &\Big(\sum_{i\in[N-T]} \lambda_1^{i-1}\mathbf{R}_{i,k}^{\mathrm{T}}, \ldots, \sum_{i\in[N-T]} \lambda_{N-T}^{i-1}\mathbf{R}_{i,k}^{\mathrm{T}}\Big)^{\mathrm{T}}\Big\}_{k\in[M+1]}, \\ &\{\mathbf{A}_{i,k}\}_{i\in\mathcal{T}, k\in\{K+1,\ldots,K+T\}} | \mathcal{M}_{l'}, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \end{aligned} \tag{165}$$

$$\leq (T+M+1)n_1n_2\Big(1 + \frac{T}{N-T}\Big)\log p \tag{166}$$

where (164) follows from (26), and (165) holds since,

$$\begin{aligned} &\Big[\textstyle\sum_{i\in[N-T]} \lambda_1^{i-1}\widetilde{\mathbf{R}}_{i,j}^{\mathrm{T}} \quad \cdots \quad \sum_{i\in[N-T]} \lambda_{N-T}^{i-1}\widetilde{\mathbf{R}}_{i,j}^{\mathrm{T}}\Big] \\ &= f(\alpha_j) - \sum_{k\in[M+1]} \Big(f(\beta_k) \\ &- \Big[\textstyle\sum_{i\in[N-T]} \lambda_1^{i-1}\mathbf{R}_{i,k}^{\mathrm{T}} \quad \cdots \quad \sum_{i\in[N-T]} \lambda_{N-T}^{i-1}\mathbf{R}_{ik}^{\mathrm{T}}\Big]^{\mathrm{T}}\Big) \\ &\prod_{k'\in[M+1]\setminus\{k\}} \frac{\alpha_j - \beta_{k'}}{\beta_k - \beta_{k'}} \end{aligned} \tag{167}$$

where the local computations $\{f(\alpha_j)\}_{j\in\mathcal{T}}$ are already known by the adversaries prior to degree reduction i.e., $\{f(\alpha_j)\}_{j\in\mathcal{T}} \in \mathcal{M}_{l'}$, and that

$$\begin{aligned} &\Big[\widetilde{\mathbf{R}}_{1,j}[s] \quad \cdots \quad \widetilde{\mathbf{R}}_{N-T,j}[s]\Big] = \\ &\Big[\textstyle\sum_{i\in[N-T]} \lambda_1^{i-1}\widetilde{\mathbf{R}}_{i,j}[s] \quad \cdots \quad \sum_{i\in[N-T]} \lambda_{N-T}^{i-1}\widetilde{\mathbf{R}}_{i,j}[s]\Big]\mathbf{M}^{-1}, \end{aligned} \tag{168}$$

for all $s \in [n_2]$, where $\widetilde{\mathbf{R}}_{i,j}[s]$ denotes the s^{th} column of $\widetilde{\mathbf{R}}_{i,j}$, and $\mathbf{M}$ is the $(N-T)\times(N-T)$ MDS matrix (hence invertible) as defined in (155). Finally, (166) holds since conditioning cannot increase entropy, and that entropy is maximized by the uniform distribution. For the second term in (163), we find that,

$$\begin{aligned} &H(\{\widetilde{\mathbf{R}}_{i,j}, \overline{\mathbf{R}}_{i,j}\}_{i\in\mathcal{H}, j\in\mathcal{T}}, \{f(\beta_k) - \mathbf{R}_k\}_{k\in[M+1]}, \{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T}, \\ k\in[M+1]}}, \\ &\qquad \{\mathbf{A}_{i,k}\}_{i\in\mathcal{T}, k\in\{K+1,\ldots,K+T\}} | \mathcal{M}_{l'}, \\ &\qquad \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in[N]}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \end{aligned}$$

$$\geq H(\{\widetilde{\mathbf{R}}_{i,j},\overline{\mathbf{R}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}},\{f(\beta_k)-\mathbf{R}_k\}_{k\in[M+1]},\{\mathbf{R}_{i,k}\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{A}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\mathcal{M}_{l'},\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]},\{f(\beta_k)\}_{k\in[M+1]}) \tag{169}$$

$$= H(\{\widetilde{\mathbf{R}}_{i,j},\overline{\mathbf{R}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}},\{\mathbf{R}_k\}_{k\in[M+1]},\{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in[M+1]}},\{\mathbf{A}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}) \tag{170}$$

$$= H\Bigg(\{\widetilde{\mathbf{R}}_{i,j},\overline{\mathbf{R}}_{i,j}\}_{\substack{i\in\mathcal{H},\\j\in\mathcal{T}}},\{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in[M+1]}},\{\mathbf{A}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in\{K+1,\ldots,K+T\}}},\Big\{\Big[\textstyle\sum_{i\in[N]}\lambda_1^{i-1}\mathbf{R}_{i,k}^{\mathrm{T}}\quad\cdots\quad\sum_{i\in[N]}\lambda_{N-T}^{i-1}\mathbf{R}_{i,k}^{\mathrm{T}}\Big]^{\mathrm{T}}\Big\}_{k\in[M+1]}\Bigg) \tag{171}$$

$$= H\Bigg(\{\widetilde{\mathbf{R}}_{i,j},\overline{\mathbf{R}}_{i,j}\}_{\substack{i\in\mathcal{H},\\j\in\mathcal{T}}},\{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in[M+1]}},\{\mathbf{A}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in\{K+1,\ldots,K+T\}}},\Big\{\Big[\sum_{i\in[N-T]}\lambda_1^{i-1}\mathbf{R}_{i,k}^{\mathrm{T}}\cdots\sum_{i\in[N-T]}\lambda_{N-T}^{i-1}\mathbf{R}_{i,k}^{\mathrm{T}}\Big]^{\mathrm{T}}\Big\}_{k\in[M+1]}\Bigg)$$

$$= H\Bigg(\{\widetilde{\mathbf{R}}_{i,j},\overline{\mathbf{R}}_{i,j}\}_{\substack{i\in\mathcal{H},\\j\in\mathcal{T}}},\{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in[M+1]}},\{\mathbf{A}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in\{K+1,\ldots,K+T\}}},\Big\{\big[\mathbf{R}_{1,k}[s]\quad\cdots\quad\mathbf{R}_{N-T,k}[s]\big]\,\mathbf{M}\Big\}_{k\in[M+1],s\in[n_2]}\Bigg) \tag{172}$$

$$= H\Bigg(\{\widetilde{\mathbf{R}}_{i,j},\overline{\mathbf{R}}_{i,j}\}_{\substack{i\in\mathcal{H},\\j\in\mathcal{T}}},\{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in[M+1]}},\{\mathbf{A}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in\{K+1,\ldots,K+T\}}},\Big\{\big[\mathbf{R}_{1,k}[s]\quad\cdots\quad\mathbf{R}_{N-T,k}[s]\big]\Big\}_{k\in[M+1],s\in[n_2]}\Bigg) \tag{173}$$

$$= H\Bigg(\{\overline{\mathbf{R}}_{i,j}\}_{\substack{i\in\mathcal{H},\\j\in\mathcal{T}}},\{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in[M+1]}},\{\mathbf{A}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in\{K+1,\ldots,K+T\}}},\Big\{\big[\mathbf{R}_{1,k}\quad\cdots\quad\mathbf{R}_{N-T,k}\big]\Big\}_{k\in[M+1]}\Bigg) \tag{174}$$

$$= H\Big(\{\overline{\mathbf{R}}_{i,j}\}_{\substack{i\in\mathcal{H},\\j\in\mathcal{T}}}|\{\mathbf{R}_{i,k}\}_{i\in[N-T],k\in[M+1]}\Big) + H(\{\mathbf{R}_{i,k}\}_{i\in[N-T],k\in[M+1]}) + H\Big(\{\mathbf{R}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in[M+1]}},\{\mathbf{A}_{i,k}\}_{\substack{i\in\mathcal{T},\\k\in\{K+1,\ldots,K+T\}}}\Big) \tag{175}$$

$$= (M+T+1)n_1n_2\Big(1+\frac{T}{N-T}\Big)\log p \tag{176}$$

where (169) holds since conditioning cannot increase entropy; (170) follows from the independence of randomness generated; (171) follows from (26); (172) follows from (155); (173) holds since $\mathbf{M}$ is a $(N-T)\times(N-T)$ MDS matrix (hence is invertible); (174) holds since $\{\widetilde{\mathbf{R}}_{i,j}\}_{j\in\mathcal{T}}$ can be perfectly reconstructed from $\{\mathbf{R}_{i,k}\}_{k\in[M+1]}$ using (20) for all $i\in\mathcal{H}$; (175) follows from the independence of the randomness generated by the honest and adversarial users; (176) follows from the entropy of uniform random variables, along with,

$$H(\{\overline{\mathbf{R}}_{i,j}\}_{\substack{i\in\mathcal{H},\\j\in\mathcal{T}}}|\{\mathbf{R}_{i,k}\}_{i\in[N-T],k\in[M+1]}) = \sum_{i\in\mathcal{H}}H\Big(\Big\{\sum_{k=K+1}^{K+T}\mathbf{A}_{i,k}\rho_{j,k}\Big\}_{j\in\mathcal{T}}\Big) \tag{177}$$

$$= \sum_{i\in\mathcal{H}}H\Big(\Big\{\big[\mathbf{A}_{i,K+1}[s]\quad\cdots\quad\mathbf{A}_{i,K+T}[s]\big]\,\mathbf{\Gamma}\Big\}_{s\in[n_2]}\Big) \tag{178}$$

$$= \sum_{i\in\mathcal{H}}H(\mathbf{A}_{i,K+1},\ldots,\mathbf{A}_{i,K+T}) \tag{179}$$

where $\rho_{j,k}$ is the Lagrange coefficient defined in (120), and $\mathbf{\Gamma}$ is a $T\times T$ MDS matrix as defined in (122) (hence is invertible).

Finally, by combining (176) and (166) with (163), we find for (161) that,

$$0\leq I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\{\widetilde{\mathbf{R}}_{i,j},\overline{\mathbf{R}}_{i,j}\}_{i\in\mathcal{H},j\in\mathcal{T}},\{f(\alpha_i)-\widetilde{\mathbf{R}}_i\}_{i\in[N]},\{\mathbf{R}_{i,k}\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{A}_{i,k}\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\mathcal{M}_{l'},\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})\leq 0 \tag{180}$$

The steps for consecutive degree reduction operations $l\in[2L+1]$ follow the same steps, from which we find for (158) and accordingly, the fourth term in (112) that,

$$\sum_{l'\in[2L+1]}I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\mathcal{M}_{4,l'}(t)|\mathcal{M}_{4,l'-1}(t)\cup\cdots\cup\mathcal{M}_{4,1}(t),\mathcal{M}_1,\mathcal{M}_2,\mathcal{M}_3,\cup_{t'=0}^{t-1}\mathcal{M}_4(t'),\cup_{t'=0}^{t-1}\mathcal{M}_5(t'),\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})$$
$$= I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\mathcal{M}_4(t)|\mathcal{M}_1,\mathcal{M}_2,\mathcal{M}_3,\cup_{t'=0}^{t-1}\mathcal{M}_4(t'),\cup_{t'=0}^{t-1}\mathcal{M}_5(t'),\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})$$
$$= 0 \tag{181}$$

E. Stage 5: Model Update

We finally consider the last term in (112), corresponding to the model update stage. In the following, we let $M=3(K+T-1)$. Then, for the last term in (112), one can write that,

$$I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\mathcal{M}_5(t)|\mathcal{M}_1,\mathcal{M}_2,\mathcal{M}_3,\cup_{t'=0}^{t}\mathcal{M}_4(t'),\cup_{t'=0}^{t-1}\mathcal{M}_5(t'),\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})$$
$$= I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T},l\in[L+1]},\{\widetilde{\mathbf{G}}_{l,i}(t)-\widetilde{\mathbf{B}}_{l,i}(t)\}_{i\in[N],l\in[L+1]},\{\mathbf{B}_{l,i,k}(t)\}_{\substack{i\in\mathcal{T},k\in[M+1],\\l\in[L+1]}},$$
$$\{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\},l\in[L+1]}|\mathcal{M}_1,\mathcal{M}_2,\mathcal{M}_3,\cup_{t'=0}^{t}\mathcal{M}_4(t'),\cup_{t'=0}^{t-1}\mathcal{M}_5(t'),\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \tag{182}$$

$$= \sum_{l\in[L+1]}I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\{\widetilde{\mathbf{G}}_{l,i}(t)-\widetilde{\mathbf{B}}_{l,i}(t)\}_{i\in[N]},\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},$$
$$\{\mathbf{S}_{l,i,k}(t)\}_{\substack{i\in\mathcal{T},\\k\in\{K+1,\ldots,K+T\}}}|\{\widetilde{\mathbf{B}}_{l',i,j}(t),\overline{\mathbf{B}}_{l',i,j}(t)\}_{\substack{i\in\mathcal{H},j\in\mathcal{T},\\l'\in[l-1]}},\{\widetilde{\mathbf{G}}_{l',i}(t)-\widetilde{\mathbf{B}}_{l',i}(t)\}_{\substack{i\in[N],\\l'\in[l-1]}},\{\mathbf{B}_{l',i,k}(t)\}_{\substack{i\in\mathcal{T},\\k\in[M+1],\\l'\in[l-1]}},$$
$$\{\mathbf{S}_{l',i,k}(t)\}_{\substack{i\in\mathcal{T},\\k\in\{K+1,\ldots,K+T\},\\l'\in[l-1]}},\mathcal{M}_1,\mathcal{M}_2,\mathcal{M}_3,\cup_{t'=0}^{t}\mathcal{M}_4(t'),\cup_{t'=0}^{t-1}\mathcal{M}_{5,t'},\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \tag{183}$$
$$= \sum_{l\in[L+1]}I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\{\widetilde{\mathbf{G}}_{l,i}(t)-\widetilde{\mathbf{B}}_{l,i}(t)\}_{i\in[N]},\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},$$

$$\{\mathbf{S}_{l,i,k}(t)\}_{\substack{i\in\mathcal{T},\\ k\in\{K+1,\ldots,K+T\}}}|\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}) \tag{184}$$

where in (184) we let,

$$\begin{aligned}\overline{\mathcal{M}}_l=(&\{\widetilde{\mathbf{B}}_{l',i,j}(t),\overline{\mathbf{B}}_{l',i,j}(t)\}_{\substack{i\in\mathcal{H},\\ j\in\mathcal{T},\\ l'\in[l-1]}},\{\widetilde{\mathbf{G}}_{l',i}(t)-\widetilde{\mathbf{B}}_{l',i}(t)\}_{\substack{i\in[N],\\ l'\in[l-1]}},\\ &\{\mathbf{B}_{l',i,k}(t)\}_{\substack{i\in\mathcal{T},\\ k\in[M+1],\\ l'\in[l-1]}},\{\mathbf{S}_{l',i,k}(t)\}_{\substack{i\in\mathcal{T},\\ k\in\{K+1,\ldots,K+T\},\\ l'\in[l-1]}},\\ &\mathcal{M}_1,\mathcal{M}_2,\mathcal{M}_3,\cup_{t'=0}^{t}\mathcal{M}_4(t'),\cup_{t'=0}^{t-1}\mathcal{M}_{5,t'})\end{aligned} \tag{185}$$

to denote the set of all messages received or generated by the adversaries prior to gradient aggregation at layer l. We next provide the analysis for a single layer $l\in[L+1]$ in (184), while noting that the same analysis holds for all layers, which can then be combined by using the chain rule in (183). For any given layer $l\in[L+1]$, one can rewrite the corresponding mutual information term from (184) as follows,

$$\begin{aligned}&I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{\substack{i\in\mathcal{H},\\ j\in\mathcal{T}}},\\ &\{\widetilde{\mathbf{G}}_{l,i}(t)-\widetilde{\mathbf{B}}_{l,i}(t)\}_{i\in[N]},\{\mathbf{B}_{l,i,k}(t)\}_{\substack{i\in\mathcal{T},\\ k\in[M+1]}},\\ &\{\mathbf{S}_{l,i,k}(t)\}_{\substack{i\in\mathcal{T},\\ k\in\{K+1,\ldots,K+T\}}}|\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})\end{aligned} \tag{186}$$

$$\begin{aligned}&=I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{\substack{i\in\mathcal{H},\\ j\in\mathcal{T}}},\\ &\{h_l(\alpha_i)-r_l(\alpha_i)\}_{i\in[N]},\{\mathbf{B}_{l,i,k}(t)\}_{\substack{i\in\mathcal{T},\\ k\in[M+1]}},\\ &\{\mathbf{S}_{l,i,k}(t)\}_{\substack{i\in\mathcal{T},\\ k\in\{K+1,\ldots,K+T\}}}|\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})\end{aligned} \tag{187}$$

$$\begin{aligned}&=I(\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{H}};\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\\ &\{h_l(\beta_k)-r_l(\beta_k)\}_{k\in[M+1]},\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},\\ &\{\mathbf{S}_{l,i,k}(t)\}_{\substack{i\in\mathcal{T},\\ k\in\{K+1,\ldots,K+T\}}}|\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})\end{aligned} \tag{188}$$

$$\begin{aligned}&=H(\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\{h_l(\beta_k)-r_l(\beta_k)\}_{k\in[M+1]},\\ &\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\\ &\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})\\ &-H(\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\{h_l(\beta_k)-r_l(\beta_k)\}_{k\in[M+1]},\\ &\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\\ &\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]})\end{aligned} \tag{189}$$

where (188) follows from the fact that there is a bijective mapping from any $M+1$ evaluation points $\{h_l(\beta_k)-r_l(\beta_k)\}_{k\in[M+1]}$ to a valid (feasible) set of local computations $\{h_l(\alpha_i)-r_l(\alpha_i)\}_{i\in[N]}$. For the first term in (189), we find that,

$$\begin{aligned}&H(\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\{h_l(\beta_k)-r_l(\beta_k)\}_{k\in[M+1]},\\ &\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\\ &\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]})\\ &=H\Big(\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\Big\{h_l(\beta_k)-\\ &\Big[\sum_{i\in[N-T]}\lambda_1^{j-1}\mathbf{B}_{l,j,k}^{\mathrm{T}}(t)\cdots\sum_{j\in[N]}\lambda_{N-T}^{j-1}\mathbf{B}_{l,j,k}^{\mathrm{T}}(t)\Big]^{\mathrm{T}}\Big\}_{k\in[M+1]},\end{aligned}$$

$$\begin{aligned}&\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\\ &\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}\Big)\end{aligned} \tag{190}$$

$$\begin{aligned}&=H\Big(\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\Big\{h_l(\beta_k)-\\ &\Big[\sum_{i\in[N-T]}\lambda_1^{i-1}\mathbf{B}_{l,i,k}^{\mathrm{T}}(t)\cdots\sum_{i\in[N-T]}\lambda_{N-T}^{i-1}\mathbf{B}_{l,i,k}^{\mathrm{T}}(t)\Big]^{\mathrm{T}}\Big\}_{k\in[M+1]},\\ &\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\\ &\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}\Big)\end{aligned} \tag{191}$$

$$\begin{aligned}&=H\Big(\{\overline{\mathbf{B}}_{l,i,j}(t)\}_{\substack{i\in\mathcal{H},\\ j\in\mathcal{T}}},\Big\{h_l(\beta_k)-\\ &\Big[\sum_{i\in[N-T]}\lambda_1^{i-1}\mathbf{B}_{l,i,k}^{\mathrm{T}}(t)\cdots\sum_{i\in[N-T]}\lambda_{N-T}^{i-1}\mathbf{B}_{l,i,k}^{\mathrm{T}}(t)\Big]^{\mathrm{T}}\Big\}_{k\in[M+1]},\\ &\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\\ &\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in\mathcal{T}},\{\mathbf{W}_l(J)\}_{l\in[L+1]}\Big)\end{aligned} \tag{192}$$

$$\leq\Big(T+(M+1)\frac{Nm}{K}\Big)\Big(1+\frac{T}{N-T}\Big)d_l d_{l-1}\log p \tag{193}$$

where the last inequality follows from the fact that uniform distribution maximizes entropy, and (192) follows from,

$$\begin{aligned}&\Big[\textstyle\sum_{i\in[N-T]}\boldsymbol{\lambda}_1^{i-1}\widetilde{\mathbf{B}}_{l,i,j}^{\mathrm{T}}(t)\quad\cdots\quad\sum_{i\in[N-T]}\boldsymbol{\lambda}_{N-T}^{i-1}\widetilde{\mathbf{B}}_{l,i,j}^{\mathrm{T}}(t)\Big]^{\mathrm{T}}\\ &=h_l(\alpha_j)-\sum_{k\in[M+1]}\Big(h_l(\beta_k)\\ &-\Big[\sum_{i\in[N-T]}\lambda_1^{i-1}\mathbf{B}_{l,i,k}^{\mathrm{T}}(t)\quad\cdots\quad\sum_{i\in[N-T]}\lambda_{N-T}^{i-1}\mathbf{B}_{l,i,k}^{\mathrm{T}}(t)\Big]^{\mathrm{T}}\Big)\\ &\prod_{k'\in[M+1]\setminus\{k\}}\frac{\alpha_j-\beta_{k'}}{\beta_k-\beta_{k'}}\end{aligned} \tag{194}$$

where $h_l(\alpha_j)$ corresponds to the local computation performed by user $j\in\mathcal{T}$, and that,

$$\begin{aligned}&\Big[\widetilde{\mathbf{B}}_{l,1,j}[s]\quad\cdots\quad\widetilde{\mathbf{B}}_{l,N-T,j}[s]\Big]=\\ &\Big[\textstyle\sum_{i\in[N-T]}\boldsymbol{\lambda}_1^{i-1}\widetilde{\mathbf{B}}_{1,j}[s]\quad\cdots\quad\sum_{i\in[N-T]}\boldsymbol{\lambda}_{N-T}^{i-1}\widetilde{\mathbf{B}}_{l,N-T,j}[s]\Big]\\ &\mathbf{M}^{-1}\end{aligned} \tag{195}$$

for all $j\in[N]$, and $s\in[d_{l-1}]$, where $\widetilde{\mathbf{B}}_{i,j}[s]$ denotes the s^{th} column of $\widetilde{\mathbf{B}}_{i,j}(t)$, and $\mathbf{M}$ is the $(N-T)\times(N-T)$ MDS matrix defined in (155) (hence invertible).

For the second term in (189), we find that,

$$\begin{aligned}&H(\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\{h_l(\beta_k)-r_l(\beta_k)\}_{k\in[M+1]},\\ &\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\\ &\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]})\\ &\geq H(\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\{h_l(\beta_k)-r_l(\beta_k)\}_{k\in[M+1]},\\ &\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\\ &\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]},\{h_l(\beta_k)\}_{k\in[M+1]})\end{aligned} \tag{196}$$

$$\begin{aligned}&=H(\{\widetilde{\mathbf{B}}_{l,i,j}(t),\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}},\{r_l(\beta_k)\}_{k\in[M+1]},\\ &\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]},\{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}|\\ &\overline{\mathcal{M}}_l,\{\mathbf{X}_i,\mathbf{Y}_i\}_{i\in[N]},\{\mathbf{W}_l(J)\}_{l\in[L+1]},\{h_l(\beta_k)\}_{k\in[M+1]})\end{aligned} \tag{197}$$

$$= H(\{\widetilde{\mathbf{B}}_{l,i,j}(t), \overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{B}_{l,k}(t)\}_{k\in[M+1]}, \{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]}, \{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}} | \overline{\mathcal{M}}_l, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in[N]}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}, \{h_l(\beta_k)\}_{k\in[M+1]}) \tag{198}$$

$$= H(\{\widetilde{\mathbf{B}}_{l,i,j}(t), \overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{B}_{l,k}(t)\}_{k\in[M+1]}, \{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]}, \{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}) \tag{199}$$

$$= H\Big(\{\widetilde{\mathbf{B}}_{l,i,j}(t), \overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]}, \{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}, \Big\{\Big[\sum_{j\in[N]}\lambda_1^{j-1}\mathbf{B}_{l,j,k}^{\mathrm{T}}(t) \cdots \sum_{j\in[N]}\lambda_{N-T}^{j-1}\mathbf{B}_{l,j,k}^{\mathrm{T}}(t)\Big]\Big\}_{k\in[M+1]}\Big) \tag{200}$$

$$= H\Big(\{\widetilde{\mathbf{B}}_{l,i,j}(t), \overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]}, \{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}, \Big\{\Big[\sum_{j=1}^{N-T}\lambda_1^{j-1}\mathbf{B}_{l,j,k}^{\mathrm{T}}(t) \cdots \sum_{j=1}^{N-T}\lambda_{N-T}^{j-1}\mathbf{B}_{l,j,k}^{\mathrm{T}}(t)\Big]\Big\}_{k\in[M+1]}\Big) \tag{201}$$

$$= H\Big(\{\widetilde{\mathbf{B}}_{l,i,j}(t), \overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]}, \{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}, \Big\{\big[\mathbf{B}_{l,1,k}[s] \quad \cdots \quad \mathbf{B}_{l,N-T,k}[s]\big]\,\mathbf{M}\Big\}_{k\in[M+1],s\in[d_{l-1}]}\Big) \tag{202}$$

$$= H\Big(\{\widetilde{\mathbf{B}}_{l,i,j}(t), \overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]}, \{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}, \Big\{\big[\mathbf{B}_{l,1,k}[s] \quad \cdots \quad \mathbf{B}_{l,N-T,k}[s]\big]\Big\}_{k\in[M+1],s\in[d_{l-1}]}\Big) \tag{203}$$

$$= H\Big(\{\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}}, \{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]}, \{\mathbf{S}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in\{K+1,\ldots,K+T\}}, \Big\{\big[\mathbf{B}_{l,1,k}(t) \quad \cdots \quad \mathbf{B}_{l,N-T,k}(t)\big]\Big\}_{k\in[M+1]}\Big) \tag{204}$$

$$= H(\{\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}} | \{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{H},k\in[M+1]}) + H(\{\mathbf{B}_{l,i,k}(t)\}_{\substack{i\in\mathcal{H},\\ k\in[M+1]}}) + H(\{\mathbf{S}_{l,i,k}(t)\}_{\substack{i\in\mathcal{T},\\ k\in\{K+1,\ldots,K+T\}}}) + H(\{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{T},k\in[M+1]}) \tag{205}$$

$$= \Big(1+\frac{T}{N-T}\Big)\Big(T+(M+1)\frac{Nm}{K}\Big)d_l d_{l-1}\log p \tag{206}$$

where the first inequality holds since conditioning cannot increase entropy; (202) holds since the $(N-T)\times(N-T)$ MDS matrix $\mathbf{M}$ defined in (155) is invertible, by letting $\mathbf{B}_{l,i,k}[s]$ denote the s^{th} column of $\mathbf{B}_{l,i,k}(t)$ for all $i \in [N-T]$; (203) holds since MDS matrices are invertible; (204) holds since each element of the random masks are generated independently; (206) follows from the fact that,

$$H(\{\overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}} | \{\mathbf{B}_{l,i,k}(t)\}_{i\in\mathcal{H},k\in[M+1]}) = \sum_{i\in\mathcal{H}} H\Big(\Big\{\sum_{k\in\{K+1,\ldots,K+T\}} \mathbf{S}_{ik}(t)\rho_{j,k}\Big\}_{j\in\mathcal{T}}\Big) \tag{207}$$

$$= \sum_{i\in\mathcal{H}} H\Big(\Big\{\big[\mathbf{S}_{i,K+1}[s] \quad \cdots \quad \mathbf{S}_{i,K+T}[s]\big]\,\mathbf{\Gamma}\Big\}_{s\in[d_{l-1}]}\Big) \tag{208}$$

where $\rho_{j,k}$ is the Lagrange coefficient defined in (120), $\mathbf{S}_{ik}[s]$ denotes the s^{th} column of $\mathbf{S}_{ik}(t)$ for $k \in \{K+1,\ldots,K+T\}$, and $\mathbf{\Gamma}$ is the $T\times T$ MDS matrix as defined in (122) (hence invertible). Finally, (206) follows from the entropy of uniform random variables.

By combining (206) and (193) with (186) and (184), we find for the last term in (112) that,

$$\begin{aligned} 0 &\le I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \mathcal{M}_5(t) | \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \cup_{t'=0}^{t}\mathcal{M}_4(t'), \cup_{t'=0}^{t-1}\mathcal{M}_{5,t'}, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &= \sum_{l\in[L+1]} I(\{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{H}}; \{\widetilde{\mathbf{B}}_{l,i,j}(t), \overline{\mathbf{B}}_{l,i,j}(t)\}_{i\in\mathcal{H},j\in\mathcal{T}}, \overline{\mathcal{M}}_l, \{\mathbf{X}_i, \mathbf{Y}_i\}_{i\in\mathcal{T}}, \{\mathbf{W}_l(J)\}_{l\in[L+1]}) \\ &= 0 \end{aligned} \tag{209}$$

which completes the proof. □

APPENDIX D
COMPUTATION COMPLEXITY

Throughout the rest of our analysis, we use the fact that interpolating a polynomial of degree κ (and evaluating it at κ points) has a computational complexity of $O(\kappa\log^2\kappa\log\log\kappa)$ [51]. We next present the per-user computation complexity of CLOVER for the individual components.

(Stage 1: Dataset Encoding) Generation of $\{\widetilde{\mathbf{X}}_{i,j}\}_{j\in[N]}$ requires evaluating a Lagrange polynomial of degree $K+T-1$ at N points, which has a complexity of $O(\frac{Ndm}{K}\log^2(K+T)\log\log(K+T))$ per user.

(Stage 2: Label Encoding) Generation of $\{\widetilde{\mathbf{Y}}_{i,j}\}_{j\in[N]}$ requires evaluating a Lagrange polynomial of degree $K+T-1$ at N points, which has a complexity of $O(\frac{Ncm}{K}\log^2(K+T)\log\log(K+T))$ per user.

(Stage 3: Model Initialization) Generation of $\{\widetilde{\mathbf{W}}_{l,i,j}(0)\}_{j\in[N]}$ requires evaluating a Lagrange polynomial of degree $K+T-1$ at N points, which has a per-user computation complexity of $O(\sum_{l\in[L+1]}\frac{Nd_l d_{l-1}}{N-T}\log^2(K+T)\log\log(K+T))$. Next, evaluating $\widetilde{\mathbf{W}}_{l,i}(0)$ has a complexity of $O(\sum_{l\in[L+1]}N(N-T)\frac{d_l d_{l-1}}{N-T})$ per user. Overall, the total computation complexity of this stage is $O(\sum_{l\in[L+1]}\frac{Nd_l d_{l-1}}{N-T}\log^2(K+T)\log\log(K+T)+\sum_{l\in[L+1]}Nd_l d_{l-1})$ per user.

(Stage 4: Gradient Computation) First, we analyze the per-user computation complexity of the degree reduction operation with DLC from Section IV, for reducing the degree of a polynomial $f(\cdot)$ of some degree $M > K+T-1$,

where $f(\beta_1), \dots f(\beta_K) \in \mathbb{F}_p^{n_1 \times n_2}$ embed the K desired computations as described in Section IV.

(Offline): The offline per-user computation cost consists of: 1) $O(n_1 n_2 \frac{N}{N-T} \log^2 M \log\log M)$ to compute the encoded random matrices $\widetilde{\mathbf{R}}_{i,j}$ in (21) for users $j \in [N]$; 2) $O(n_1 n_2 \frac{N}{N-T} \log^2(K+T) \log\log(K+T))$ for computing the encoded matrices $\overline{\mathbf{R}}_{i,j}$ from (23) for users $j \in [N]$; 3) $O(N(N-T)\frac{n_1 n_2}{N-T})$ for evaluating $\widetilde{\mathbf{R}}_i$ from (24); 4) $O(N(N-T)\frac{n_1 n_2}{N-T})$ to evaluate $\overline{\mathbf{R}}_i$ from (25). Overall, the offline overhead is $O(\frac{N}{N-T} n_1 n_2 \log^2(K+T) \log\log(K+T) + N n_1 n_2)$ per user.

(Online): The online per-user computation cost consists of: 1) $O(n_1 n_2)$ for computing $f(\alpha_i) - \widetilde{\mathbf{R}}_i$ from (29); 2) $O(n_1 n_2 M \log^2 M \log\log M)$ for interpolating the degree M polynomial $\varphi(\alpha)$ from (30); $O(n_1 n_2 K)$ for the re-encoding operation from (32). Overall, the online computation overhead is $O(n_1 n_2 M \log^2 M \log\log M)$ per user.

As a result, the total per-user computation overhead of degree reduction with DLC, including both online and online phases, is given by,

$$O\Bigg(n_1 n_2 \frac{N}{N-T} \log^2 M \log\log M + n_1 n_2 M \log^2 M \log\log M + n_1 n_2 N\Bigg) \quad (210)$$

We next use (210) to analyze the computation overhead for gradient computation. At each training round $t \in \{0, \dots, J-1\}$, the per-user computation overhead for gradient computation consists of the following components.

Forward Propagation: The per-user computation overhead of forward propagation consists of: 1) $O(\frac{Nm}{K} \sum_{l\in[L+1]} d_l d_{l-1})$ to compute $\widetilde{\mathbf{Z}}_{l,i}(t)$ from (62); 2) $O\Big(\sum_{l\in[L+1]} d_l \frac{Nm}{K}(\frac{N}{N-T} + K + T) \log^2(K+T) \log\log(K+T) + \sum_{l\in[L+1]} d_l \frac{Nm}{K} N\Big)$ for the degree reduction operation from (63), by letting $M = 3(K+T-1)$, $n_1 = d_l$, and $n_2 = \frac{Nm}{K}$ in (210); 3) $O(\frac{Nm}{K} \sum_{l\in[L+1]} d_l)$ for computing $\widetilde{\mathbf{U}}_{l,i}(t)$ from (65). Hence, the overall per-user computation overhead for forward propagation is given as,

$$O\Bigg(\frac{Nm}{K}\Big(\sum_{l\in[L+1]} d_l d_{l-1} + \sum_{l\in[L+1]} d_l(\frac{N}{N-T} + K + T) \log^2(K+T) \log\log(K+T) + \sum_{l\in[L+1]} N d_l\Big)\Bigg) \quad (211)$$

Backpropagation: The per-user computation overhead of the backpropagation operation consists of the following: 1) $O\Big(\frac{Nm}{K}(c + \sum_{l\in[L]} d_l d_{l+1})\Big)$ to compute the error term $\widetilde{\mathbf{E}}_{l,i}(t)$ from (66); 2) $O\Big(\frac{Nmd_l}{K}(\frac{N}{N-T} + K + T) \log^2(K+T) \log\log(K+T) + \sum_{l\in[L+1]} \frac{N^2 m d_l}{K}\Big)$ for the degree reduction operation from (67), by letting $M = 3(K+T-1)$, $n_1 = d_l$, and $n_2 = \frac{Nm}{K}$ in (210); 3) $O(\frac{Nm}{K} \sum_{l\in[L+1]} d_l d_{l-1})$ for computing the gradient from (68). Then, the overall per-user computation overhead for backpropagation is,

$$O\Bigg(\frac{Nm}{K}\Big(\sum_{l\in[L+1]} d_l d_{l-1} + \sum_{l\in[L+1]} d_l(\frac{N}{N-T} + K + T) \log^2(K+T) \log\log(K+T) + \sum_{l\in[L+1]} N d_l\Big)\Bigg) \quad (212)$$

Finally, the total per-user computation overhead for gradient computation, including both forward propagation and backpropagation is,

$$O\Bigg(\frac{Nm}{K}\Big(\sum_{l\in[L+1]} d_l d_{l-1} + \sum_{l\in[L+1]} d_l(\frac{N}{N-T} + K + T) \log^2(K+T) \log\log(K+T) + \sum_{l\in[L+1]} N d_l\Big)\Bigg) \quad (213)$$

(Stage 5: Gradient Aggregation and Model Update) At each training round $t \in \{0, \dots, J-1\}$, the per-user computation overhead for model updating consists of the following components.

(Offline): The offline per-user computation cost consists of: 1) $O(\sum_{l\in[L+1]} \frac{d_l d_{l-1}}{N-T} N \log^2(K+T) \log\log(K+T))$ for evaluating $\widetilde{\mathbf{B}}_{l,i,j}$ for all $j \in [N]$; 2) $O(\sum_{l\in[L+1]} (N-T) N \frac{d_l d_{l-1}}{N-T})$ to compute $\widetilde{\mathbf{B}}_{l,i}(t)$ in (71); 3) $O\big(\sum_{l\in[L+1]} \frac{d_l d_{l-1}}{N-T} N \log^2(K+T) \log\log(K+T) + \sum_{l\in[L+1]} \frac{d_l d_{l-1}}{N-T} K\big)$ for evaluating $\overline{\mathbf{B}}_{i,j}(t)$ for all $j \in [N]$, where the last term is due to evaluating the sum $\sum_{k'\in[K]} \mathbf{B}_{l,i,k'}(t)$ in (70); 4) $O(\sum_{l\in[L+1]} N d_l d_{l-1})$ for computing $\overline{\mathbf{B}}_{l,i}(t)$.

(Online): The online per-user computation cost consists of: 1) $O(\sum_{l\in[L+1]} d_l d_{l-1})$ for evaluating $\widehat{\mathbf{G}}_{l,i}(t)$ from (77); 2) $O(\sum_{l\in[L+1]} d_l d_{l-1}(K+T) \log^2(K+T) \log\log(K+T))$ for interpolating the degree $3(K+T-1)$ polynomial $h_l(\alpha) - r_l(\alpha)$ and evaluating the masked gradients $h_l(\beta_k) - r_l(\beta_k)$ from (79) for all $k \in [K]$; 3) $O(\sum_{l\in[L+1]} K d_l d_{l-1})$ for evaluating the aggregated gradient $\widetilde{\mathbf{G}}_{l,i}(t)$ from (80); 4) $O(\sum_{l\in[L+1]} d_l d_{l-1})$ for the model update.

As a result, the total per-user computation overhead for model updating, including both online and online phases, is given as,

$$O\Bigg(\sum_{l\in[L+1]} \frac{d_l d_{l-1}}{N-T} \log^2(K+T) \log\log(K+T) + \sum_{l\in[L+1]} N d_l d_{l-1} + \sum_{l\in[L+1]} d_l d_{l-1}(K+T) \log^2(K+T) \log\log(K+T) + \sum_{l\in[L+1]} K d_l d_{l-1}\Bigg) \quad (214)$$

$$= O\Bigg(\sum_{l\in[L+1]} d_l d_{l-1}\Big(\frac{N}{N-T} + K + T\Big) \log^2(K+T) \log\log(K+T) + \sum_{l\in[L+1]} d_l d_{l-1} N\Bigg) \quad (215)$$

Finally, by combining all five stages, for J training rounds, the total per-user computation complexity is,

$$O\Bigg((d+c)\frac{Nm}{K}\log^2(K+T)\log\log(K+T) + J\Bigg(\sum_{l\in[L]}\frac{Nm}{K}d_l d_{l-1} + \sum_{l\in[L+1]}\frac{N^2m}{K}d_l + \sum_{l\in[L+1]}d_l\Big(d_{l-1} \tag{216}$$

$$+\frac{Nm}{K}\Big)\Big(\frac{N}{N-T}+K+T\Big)\log^2(K+T)\log\log(K+T)\Big)\Bigg) \tag{217}$$

Appendix E
Details of the Model Updating Process

For tractability of the theoretical analysis, in our privacy analysis from Section VII we consider a sufficiently large field size and treat all training operations as finite field polynomial operations. In our experiments, we instead leverage the secure truncation mechanism described in Section VIII for the model update in (83), to reduce the required field size in practice, albeit with a slight loss in accuracy. In this section, we provide the details on how one can represent all training computations using finite field polynomial operations only, consisting of finite field addition and multiplications, where the main challenge is the fact that that $\frac{\eta}{Nm} \ll 1$ in (5). To do so, one can select a learning rate η such that $\delta \triangleq \frac{Nm}{\eta} \in \mathbf{Z}_+$, which is then mapped to the finite field $\mathbb{F}_p$ as described in App. E. Then, the intended model update from (5) is given as,

$$\begin{aligned}\mathbf{W}_l(t+1) &= \mathbf{W}_l(t) - \frac{\eta}{Nm}\mathbf{G}_l(t)\\ &= \mathbf{W}_l(t) - \frac{1}{\delta}\mathbf{G}_l(t) \quad \forall l\in[L+1]\end{aligned} \tag{218}$$

In the following, we show that the intended model $\{\mathbf{W}_l(t+1)\}_{l\in[L+1]}$ from (218) can be obtained using finite field polynomial operations only, at any training round $t\in\{1,\ldots,J-1\}$.

Proposition 1: The model $\mathbf{W}_1(t+1),\ldots,\mathbf{W}_{L+1}(t+1)$ from (218) can be obtained using finite field polynomial operations only, by defining a new error propagation rule that replaces (66) as,

$$\widetilde{\mathbf{E}}_{l,i}(t) = \begin{cases} 2(\widetilde{\mathbf{Z}}_{L+1,i}(t) - \delta_t^{2^{L+1}-1}\widetilde{\mathbf{Y}}_i(t)) & \text{if} \quad l=L+1\\ 2\widetilde{\mathbf{Z}}_{l,i}(t)\odot(\widetilde{\mathbf{W}}_{l+1,i}^{\mathrm{T}}(t)\times\widetilde{\mathbf{E}}_{l+1,i}(t)) & \text{if} \quad l\le L\end{cases} \tag{219}$$

and a model update rule that replaces (83) as,

$$\widetilde{\mathbf{W}}_{l,i}(t+1) = \delta\times\delta_t^{3\times 2^{L+1}-3}\widetilde{\mathbf{W}}_{l,i}(t) - \delta_t^{2^{L+1}+1}\widetilde{\mathbf{G}}_{l,i}(t) \tag{220}$$

where

$$\delta_t \triangleq \begin{cases}1 & \text{if} \quad t=0\\ \delta\times\delta_{t-1}^{3\times 2^{L+1}-2} & \text{if} \quad t>0\end{cases} \tag{221}$$

At any training round $t\in\{0,\ldots,J-1\}$, users can recover the intended model from (218) via polynomial interpolation, by collecting the local computations $\widetilde{\mathbf{W}}_{l,i}(t+1)$ in (220) from any set of at least $(K+T-1)+1$ users, and then re-scaling the decoded model as,

$$\mathbf{W}_l(t+1) \leftarrow \frac{\mathbf{W}_l(t+1)}{\delta_{t+1}} \tag{222}$$

Proof: We next show that using the model update rule from (220), one can recover the intended model from (218) at any round $t\in\{0,\ldots,J-1\}$. We first define a virtual variable,

$$\overline{\mathbf{W}}_l(t+1) = \delta\times\delta_t^{3\times 2^{L+1}-3}\overline{\mathbf{W}}_l(t) - \delta_t^{2^{L+1}+1}\overline{\mathbf{G}}_l(t) \quad \forall l\in[L+1] \tag{223}$$

where $\overline{\mathbf{W}}_l(0) \triangleq \mathbf{W}_l(0)$, $\overline{\mathbf{G}}_l(0) \triangleq \mathbf{G}_l(0)$, and $\overline{\mathbf{W}}_l(t+1)$ corresponds to the model obtained if the encoded model $\widetilde{\mathbf{W}}_{l,i}(t+1)$ from (220) is decoded at the end of round t, by collecting $\widetilde{\mathbf{W}}_{l,i}(t+1)$ from at any set of at least $(K+T-1)+1$ users and using polynomial interpolation. Accordingly, $\overline{\mathbf{G}}_l(t)$ corresponds to the aggregated gradient obtained if the encoded gradient $\widetilde{\mathbf{G}}_{l,i}(t)$ from (220) is decoded using polynomial interpolation at round t, by collecting $\widetilde{\mathbf{G}}_{l,i}(t)$ from any set of at least $(K+T-1)+1$ users, and then aggregating the resulting gradients. We next show that,

$$\mathbf{W}_l(t+1) = \frac{\overline{\mathbf{W}}_l(t+1)}{\delta_{t+1}} \quad \forall t\in\{0,\ldots,J-1\}, \tag{224}$$

hence the model update operation from (220) can perfectly recover the intended model $\mathbf{W}_l(t+1)$ from (218). The proof then follows by induction, where we provide the details next.

1. Base case ($t=0$). For the base case, we observe from (220) and (221) that,

$$\overline{\mathbf{W}}_l(1) = \delta\overline{\mathbf{W}}_l(0) - \overline{\mathbf{G}}_l(0) \quad \forall l\in[L+1], \tag{225}$$

hence (224) holds for the base case $t=0$,

$$\frac{\overline{\mathbf{W}}_l(1)}{\delta_1} = \frac{\overline{\mathbf{W}}_l(1)}{\delta} = \overline{\mathbf{W}}_l(0) - \frac{1}{\delta}\overline{\mathbf{G}}_l(0) \quad \forall l\in[L+1] \tag{226}$$

2. Induction step ($t>0$). Next, we assume that (224) holds for an arbitrary round $t-1$, and show that it also holds for round t. We first note that,

$$\overline{\mathbf{W}}_l(t) = \delta_t\mathbf{W}_l(t) \quad \forall l\in[L+1], \tag{227}$$

since (224) holds by assumption at round $t-1$, and then evaluate the aggregated gradient $\{\overline{\mathbf{G}}_l(t)\}_{l\in[L+1]}$ at round t from the forward and backpropagation of $\{\overline{\mathbf{W}}_l(t)\}_{l\in[L+1]}$. For forward propagation, let $\overline{\mathbf{Z}}_l(t)$ and $\overline{\mathbf{U}}_l(t)$ denote the signal at layer $l\in[L+1]$ before and after the activation function $g(\cdot)$, corresponding to the forward pass of the model $\{\overline{\mathbf{W}}_l(t)\}_{l\in[L+1]}$, hence $\overline{\mathbf{U}}_l(t) = g(\overline{\mathbf{Z}}_l(t))$ and $\overline{\mathbf{Z}}_l(t) = \overline{\mathbf{W}}_l(t)\overline{\mathbf{U}}_{l-1}(t)$. Let $\mathbf{Z}_l(t)$ and $\mathbf{U}_l(t)$ correspond to the forward pass of the true model $\{\mathbf{W}_l(t)\}_{l\in[L+1]}$ before and after the activation function $g(\cdot)$ at layer $l\in[L+1]$, hence $\mathbf{U}_l(t) = g(\mathbf{Z}_l(t))$ and $\mathbf{Z}_l(t) = \mathbf{W}_l(t)\mathbf{U}_{l-1}(t)$. From (227), we observe that,

$$\overline{\mathbf{Z}}_l(t) = \delta_t^{\sum_{l'=1}^{l}2^{l'}-1}\mathbf{Z}_l(t) = \delta_t^{2^l-1}\mathbf{Z}_l(t) \quad \forall l\in[L+1] \tag{228}$$

and

$$\overline{\mathbf{U}}_l(t) = \delta_t^{\sum_{l'=1}^{l}2^{l'}}\mathbf{U}_l(t) = \delta_t^{2(2^l-1)}\mathbf{U}_l(t) \quad \forall l\in[L+1] \tag{229}$$

Next, for backpropagation, let $\overline{\mathbf{E}}_l(t)$ denote the error at layer l obtained from the error propagation operation from (219) for the virtual model $\{\overline{\mathbf{W}}_l(t)\}_{l\in[L+1]}$,

$$\overline{\mathbf{E}}_l(t) = \begin{cases} 2(\overline{\mathbf{Z}}_{L+1}(t) - \delta_t^{2^{L+1}-1}\mathbf{Y}(t)) & \text{if} \quad l = L+1 \\ 2\overline{\mathbf{Z}}_l(t) \odot (\overline{\mathbf{W}}_{l+1}^{\mathrm{T}}(t) \times \overline{\mathbf{E}}_{l+1}(t)) & \text{if} \quad l \leq L \end{cases} \tag{230}$$

and $\mathbf{E}_l(t)$ denote the error propagation operation from (66) for the true model $\{\mathbf{W}_l(t)\}_{l\in[L+1]}$,

$$\mathbf{E}_l(t) = \begin{cases} 2(\mathbf{Z}_{L+1}(t) - \mathbf{Y}(t)) & \text{if} \quad l = L+1 \\ 2\mathbf{Z}_l(t) \odot (\mathbf{W}_{l+1}^{\mathrm{T}}(t) \times \mathbf{E}_{l+1}(t)) & \text{if} \quad l \leq L \end{cases} \tag{231}$$

From (227), (230), and (231), one can observe that,

$$\overline{\mathbf{E}}_l(t) = \delta_t^{(\sum_{l'=l}^{L+1} 2^{l'})-1}\mathbf{E}_l(t) = \delta_t^{2^{L+2}-2^l-1}\mathbf{E}_l(t) \quad \forall l \in [L+1] \tag{232}$$

Then, from (229) and (232), we observe the following relationship between the gradient $\overline{\mathbf{G}}_l(t) = \overline{\mathbf{E}}_l(t) \times \overline{\mathbf{U}}_{l-1}^{\mathrm{T}}(t)$ corresponding to the virtual model $\overline{\mathbf{W}}_l(t)$, and the gradient $\mathbf{G}_l(t) = \mathbf{E}_l(t) \times \mathbf{U}_{l-1}^{\mathrm{T}}(t)$ corresponding to the original model $\mathbf{W}_l(t)$,

$$\overline{\mathbf{G}}_l(t) = \delta_t^{2^{L+2}-3}\mathbf{G}_l(t) \quad \forall l \in [L+1] \tag{233}$$

from which we find that,

$$\frac{\overline{\mathbf{W}}_l(t+1)}{\delta_{t+1}} = \frac{\delta \times \delta_t^{3\times 2^{L+1}-3}\overline{\mathbf{W}}_l(t) - \delta_t^{2^{L+1}+1}\overline{\mathbf{G}}_l(t)}{\delta \times \delta_t^{3\times 2^{L+1}-2}} \tag{234}$$

$$= \mathbf{W}_l(t) - \frac{1}{\delta}\mathbf{G}_l(t) \tag{235}$$

$$= \mathbf{W}_l(t+1) \tag{236}$$

where (235) follows from (227) and (233), respectively, which completes the proof. □

REFERENCES

[1] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Security Privacy (SP)*, Jul. 2017, pp. 19–38.

[2] M. Ben-Or and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proc. 20th Annu. ACM Symp. Theory Comput. (STOC)*, 1988, pp. 1–10.

[3] I. Damgård and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," in *Proc. Annu. Int. Cryptol. Conf.*, 2007, pp. 572–590.

[4] Z. Beerliová-Trubìniová and M. Hirt, "Perfectly-secure MPC with linear communication complexity," in *Proc. Theory Cryptogr. Conf.*, 2008, pp. 213–230.

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

[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 *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 334–348.

[7] A. Gascón et al., "Privacy-preserving distributed linear regression on high-dimensional data," *Proc. Privacy Enhancing Technol.*, vol. 2017, no. 4, pp. 345–364, Oct. 2017.

[8] P. Mohassel and P. Rindal, "ABY 3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 35–52.

[9] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: Efficient and private neural network training," *IACR Cryptol. ePrint Arch.*, vol. 442, 2018. [Online]. Available: https://eprint.iacr.org/2018/442

[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 *Proc. 22nd Int. Conf. Artif. Intell. Statist.*, 2019, pp. 1215–1225.

[11] J. So, B. Güler, and S. Avestimehr, "A scalable approach for privacy-preserving collaborative machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2020, pp. 1–24.

[12] J. So, B. Güler, and A. S. Avestimehr, "CodedPrivateML: A fast and privacy-preserving framework for distributed machine learning," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 1, pp. 441–451, Mar. 2021.

[13] S. Dutta, Z. Bai, H. Jeong, T. Meng Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized PolyDot codes for matrix multiplication," 2018, *arXiv:1811.10751*.

[14] M. Aliasgari, O. Simeone, and J. Kliewer, "Private and secure distributed matrix multiplication with flexible communication load," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2722–2734, 2020.

[15] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[16] K. Bonawitz et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1–24.

[17] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1253–1269.

[18] J. So et al., "Lightsecagg: A lightweight and versatile design for secure aggregation in federated learning," *Proc. Mach. Learn. Syst.*, vol. 4, pp. 694–720, May 2022.

[19] Y. Zhao and H. Sun, "Information theoretic secure aggregation with user dropouts," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2021, pp. 1124–1129.

[20] A. R. Elkordy, J. Zhang, Y. H. Ezzeldin, K. Psounis, and S. Avestimehr, "How much privacy does federated learning with secure aggregation guarantee?" *Proc. Privacy Enhancing Technol.*, vol. 2023, no. 1, pp. 510–526, Jan. 2023.

[21] J. So, R. E. Ali, B. Guler, J. Jiao, and S. Avestimehr, "Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 1–24.

[22] M. Lam, G.-Y. Wei, D. Brooks, V. Reddi, and M. Mitzenmacher, "Gradient disaggregation: Breaking privacy in federated learning by reconstructing the user participant matrix," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 1–22.

[23] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proc. Theory Cryptogr. Conf.*, 2006, pp. 265–284.

[24] S. Samet, "Privacy-preserving logistic regression," *J. Adv. Inf. Technol.*, vol. 1, no. 1, pp. 88–95, May 2015.

[25] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 1310–1321.

[26] M. Abadi et al., "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 308–318.

[27] M. Pathak, S. Rane, and B. Raj, "Multiparty differential privacy via aggregation of locally trained classifiers," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2010, pp. 1876–1884.

[28] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–24.

[29] A. Rajkumar and S. Agarwal, "A differentially private stochastic gradient descent algorithm for multiparty classification," in *Proc. 15th Int. Conf. Artif. Intell. Statist.*, 2012, pp. 933–941.

[30] B. Jayaraman, L. Wang, D. Evans, and Q. Gu, "Distributed learning without distress: Privacy-preserving empirical risk minimization," in *Proc. Adv. in Neural Inf. Process. Syst.*, 2018, pp. 6346–6357.

[31] W.-N. Chen, A. Ozgur, and P. Kairouz, "The Poisson binomial mechanism for unbiased federated learning with secure aggregation," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 3490–3506.

[32] 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 *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 3056–3089.

[33] P. Kairouz, Z. Liu, and T. Steinke, "The distributed discrete Gaussian mechanism for federated learning with secure aggregation," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 5201–5212.

[34] C. Gentry and D. Boneh, *A Fully Homomorphic Encryption Scheme*, vol. 20. Stanford, CA, USA: Stanford Univ. Press, 2009.

[35] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput. (STOC)*, 2009, pp. 168–178.
[36] 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 *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.
[37] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," 2017, *arXiv:1711.05189*.
[38] T. Graepel, K. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," in *Proc. Int. Conf. Inf. Secur. Cryptol.*, 2012, pp. 1–21.
[39] J. Yuan and S. Yu, "Privacy preserving back-propagation neural network learning made practical with cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 212–221, Jan. 2014.
[40] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *IACR Cryptol. ePrint Arch.*, vol. 1, p. 35, Jul. 2017.
[41] P. Li, J. Li, Z. Huang, C.-Z. Gao, W.-B. Chen, and K. Chen, "Privacy-preserving outsourced classification in cloud computing," *Cluster Comput.*, vol. 21, no. 1, pp. 277–286, Mar. 2018.
[42] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon, "Logistic regression model training based on the approximate homomorphic encryption," *BMC Med. Genomics*, vol. 11, no. S4, p. 83, Oct. 2018.
[43] Q. Wang et al., "Privacy-preserving collaborative model learning: The case of word vector training," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 12, pp. 2381–2393, Dec. 2018.
[44] K. Han, S. Hong, J. H. Cheon, and D. Park, "Logistic regression on homomorphic encrypted data at scale," in *Proc. Annual Conf. Innov. App. Artif. Intell. (IAAI)*, 2019, pp. 1–27.
[45] R. Livni, S. Shalev-Shwartz, and O. Shamir, "On the computational efficiency of training neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, pp. 1–27.
[46] E. W. Cheney and W. A. Light, *A Course in Approximation Theory*, vol. 101. Washington, DC, USA: American Mathematical Soc., 2009.
[47] M. H. Stone, "The generalized weierstrass approximation theorem," *Math. Mag.*, vol. 21, no. 4, p. 167, Mar. 1948.
[48] X. Lu, H. U. Sami, and B. Güler, "Privacy-preserving collaborative learning with linear communication complexity," *IEEE Trans. Inf. Theory*, vol. 70, no. 8, pp. 5857–5887, Aug. 2024.
[49] 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 *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 1–11.
[50] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *Proc. Int. Conf. Financial Cryptogr. Data Secur. (FC)*, Tenerife, Spain, Jan. 2010, pp. 35–50.
[51] K. S. Kedlaya and C. Umans, "Fast polynomial factorization and modular composition," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1767–1802, 2011.
[52] Y. LeCun, C. Cortes, and C. Burges. (2010). *MNIST Handwritten Digit Database*. [Online]. Available: http://yann.lecun.com/exdb/mnist
[53] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," M.S. thesis, Citeseer, Dept. Comput. Sci., Univ. Toronto, 2009.
[54] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[55] G. G. Chrysos, S. Moschoglou, G. Bouritsas, J. Deng, Y. Panagakis, and S. Zafeiriou, "Deep polynomial neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 8, pp. 4021–4034, Aug. 2022.
[56] G. G. Chrysos, S. Moschoglou, G. Bouritsas, Y. Panagakis, J. Deng, and S. Zafeiriou, "P–nets: Deep polynomial neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 7325–7335.
[57] G. G. Chrysos, M. Georgopoulos, J. Deng, J. Kossaifi, Y. Panagakis, and A. Anandkumar, "Augmenting deep classifiers with polynomial neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2022, pp. 692–716.
[58] A. Dubey, F. Radenovic, and D. Mahajan, "Scalable interpretability via polynomials," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 36748–36761.
[59] E. A. Rocamora, M. F. Sahin, F. Liu, G. Chrysos, and V. Cevher, "Sound and complete verification of polynomial networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 3517–3529.
[60] Y. Cheng, G. G. Chrysos, M. Georgopoulos, and V. Cevher, "Multi-linear operator networks," in *Proc. Int. Conf. Learn. Represent.*, 2024, pp. 1–24.
[61] J.-K. Zinzindohoué, K. Bhargavan, J. Protzenko, and B. Beurdouche, "HACL: A verified modern cryptographic library," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1789–1806.
[62] T. Granlund. (2004). *GNU MP: The GNU Multiple Precision Arithmetic Library*. [Online]. Available: http://gmplib.org/
[63] J. So, B. Güler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 2168–2181, Jul. 2021.
[64] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Proc. 28th Annu. Symp. Found. Comput. Sci. (SFCS)*, 1987, pp. 427–438.
[65] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2017, pp. 119–129.
[66] M. Fahim and V. R. Cadambe, "Numerically stable polynomially coded computing," *IEEE Trans. Inf. Theory*, vol. 67, no. 5, pp. 2758–2785, May 2021.

Xingyu Lu received the Bachelor of Engineering degree from the Computer Science and Information Technology Department, Zhejiang Gongshang University, China, in 2019, and the Master of Science degree in robotics (computer science) from the Khoury College of Computer Science and the College of Engineering, Northeastern University, Boston, MA, USA, in 2021. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, University of California at Riverside, Riverside. His research interests include private machine learning, distributed learning, and federated learning.

Umit Yigit Basaran received the B.Sc. degree in computer science from Ihsan Dogramaci Bilkent University, Ankara, Türkiye, in 2022. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of California at Riverside, Riverside. His research interests include federated and distributed machine learning, secure and private computing, machine unlearning, and information theory.

Başak Güler (Member, IEEE) received the B.Sc. degree in electrical and electronics engineering from Middle East Technical University (METU), Ankara, Türkiye, and the Ph.D. degree from the Wireless Communications and Networking Laboratory, The Pennsylvania State University, in 2017. From 2018 to 2020, she was a Post-Doctoral Scholar with the University of Southern California. She is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of California at Riverside, Riverside. Her research interests include information theory, distributed computing, machine learning, and wireless networks. She received the NSF CAREER Award in 2022 and serves as an Associate Editor for IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, in the area of artificial intelligence and machine learning, and IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, in the area of green computing and artificial intelligence.