DeepPrivate: Scalable Distributed DNN Training with Data and Model Privacy

Nishant Relan[1], Yihan Jiang[1], Songze Li[2], Robert Raynor[1], and Sreeram Kannan[1]
[1] University of Washington
[2] Hong Kong University of Science and Technology

***Abstract*—A key friction in the information era is between the high utility of data and stringent privacy considerations. One possible resolution is to perform distributed learning with privacy guarantee. Consider N nodes, each of which stores locally some (function of) data from a set of clients, collaboratively train a model that is requested by a remote learner node, using data from all clients collectively. For this task, we develop DeepPrivate, a protocol that provides information-theoretic guarantees *simultaneously* for data and model privacy against any $T < N$ colluding workers and data privacy against the curious learner. To achieve this, DeepPrivate partitions the entire dataset into $K < N$ parts, and leverages secret sharing and compatible erasure coding to securely distribute the coded data partitions and intermediate computation results during the training process. As a result, DeepPrivate achieves a factor $K = \Theta(N)$ reduction in computation complexity compared with state-of-the-art private machine-learning frameworks. Furthermore, DeepPrivate improves the scalability and security via layer-by-layer training, which allows the workers to only compute polynomials of low degree during training, at a small cost on the model accuracy.**

I. Introduction

Data privacy and security have become a significant consideration for modern data analytics. Therefore, machine learning methods that achieve high accuracy while preserving privacy are of great interest. In this paper we consider a distributed learning system consisting of N worker nodes (e.g., edge servers), each of whom stores some data from a set of clients (e.g., smartphones and IoT devices), and a remote learner node who wants to obtain a deep neural network (DNN) model trained using all clients' data (Figure 1). The training process has to be performed such that 1) the clients' data has to be kept private from curious (potentially colluding) worker nodes and the curious learner; and 2) the final trained model has to be kept private from the workers. The above model fits well with a wide range of applications. Two typical examples are given below.

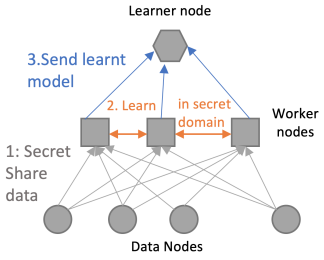

Fig. 1. System Model

1. Secure data market. A group of N worker nodes collaborate to provide the functionality of a data market, on which the knowledge extracted from data, rather than the data itself, is traded. For instance, data owners store their data in a privacy-preserving manner on the market, which can be purchased to contribute to training machine learning models intended by some learner on the other side of the market. During this trading process, the market should not gain any knowledge about the data and the model, and the learner should not know anything about the data other than what can be inferred from the trained model.

2. Secure cross-silo federated learning. In this case, a learner node (which may host some paid data-analytics services) would like to extract relevant knowledge from participating institutes (e.g., banks or hospitals) whose data are highly confidential. In this case, the data owners and the worker nodes can co-locate at some institutes. Importantly, by the end of the training process, the learner should be the only party that possesses the trained model, as opposed to the learning participants.

Currently, privacy-preserving DNN training is achieved by techniques of homomorphic encryption (HE) (see, e.g., [3, 19, 24]) and secure multiparty computation (MPC) (see, e.g., [1, 33, 34, 46]). HE-based algorithms perform DNN training and inference on cryptographically encrypted data, which requires approximating the activation functions in a neural network as low-degree polynomials (as HE only works with addition and multiplication), and thus compromising the accuracy of the trained model. For the MPC-based schemes, secret shares of the original data are distributed to different parties, which are used locally to train the network in the secret domain. While MPC protocols have been developed for secure DNN training in two-party [1, 34] and three-party [33, 46] settings, they do not naturally extend to general N-party cases. For all of the developed HE and MPC based schemes, each compute node operates on an encrypted or shared data that is of the same size of the original data, and hence the computation efficiency (e.g., training time) does not scale with the number of compute nodes. Both HE and MPC based schemes focus on protecting data privacy. For a similar setting where a set of clients, each of which has some local data, collaborate with a remote server to train a global model, federated learning [31] has recently emerged as a secure distributed ML paradigm where each client trains a local model using its local data, and communicates its learnt model (but not data) to the server. The server aggregates the received local models to generate and update a global model. Various techniques involving differential privacy [18, 32, 45] and secure aggregation [10] have been developed to protect the data privacy of the clients from the local models. However, little attention has been paid on protecting the privacy of the

global model against curious clients.

Recently, a decentralized secure machine learning framework named COPML is proposed in [42] for logistic regression problem. COPML reduces the computation complexity at each worker node compared with conventional MPC schemes, and achieves information-theoretic data privacy. Specifically, COPML partitions the entire dataset into $K < N$ parts, and applies another layer of Lagrange encoding [50] on top of Shamir secret sharing [39], to create N coded data partitions, each of which has a size of $1/K$ of the entire dataset and processed at a single node. In this way, the computation load of each node is reduced by a factor of K, while the data privacy is still protected by Lagrange coding.

We propose DeepPrivate to further extend COPML to train deep neural networks. A key enabler of DeepPrivate is to adopt layer-by-layer training (see e.g., [8]), such that a neural network is trained one layer at a time. We experimentally demonstrate on various datasets that layer-by-layer training, when combined with quantization, polynomial activation and loss functions, incurs negligible accuracy loss compared with models learnt end-to-end using ReLU activation and cross-entropy loss.

This allows each DeepPrivate node to compute a polynomial with a small degree in training a single layer, and apply COPML to achieve data privacy and improve computation efficiency. By the end of training a single layer, each worker has only a secret share of the trained model, satisfying the model privacy at the clients. Moreover, workers add additional randomness to secret shares forwarded to the remote learner, guaranteeing a full reconstruction of the model in the clear without leaking any additional information about clients' data. Finally, DeepPrivate maintains the data privacy at the worker nodes across layers by having each worker compute a secret share of the input to the next layer by passing a share of the input to the current layer through a share of the just trained layer parameters.

To summarize, our main contribution is reflected in the following salient features of DeepPrivate:

- Significant improvement on computation complexity compared with conventional HE and MPC based secure DNN training algorithms;
- Simple implementation based on layer-by-layer training for DNNs. Low complexity for (coded) gradient computation and decoding at each client.
- Does not require any cryptographic primitives. Essentially only Reed-Solomon codes are needed.
- Formal information-theoretic security for data and model at colluding workers, and for data at the learner.

Related works

Homomorphic encryption (HE) [17] techniques have been leveraged to perform privacy-preserving machine learning [2, 11, 19–21, 23, 24, 29, 43, 51]. Specifically, a client uploads a ciphertext of its data to a remote server who performs training or inference directly over the encrypted data. This approach works for DNNs by approximating the activation functions (e.g., sigmoid or ReLU) and the max pooling operation using low-degree polynomials to achieve reasonable computation throughput. While HE-based approaches enjoy small communication overhead between the client and the server, they often suffer from implementing complicated cryptographic tools like public-key encryption/decryption, and accuracy loss due to polynomial approximations.

Apart from HE, protocols based on multiparty computation (MPC) [7, 14, 30] have been proposed for secure machine learning tasks [13, 30, 36], and particularly for deep neural networks (see, e.g., [1, 34, 46]). In this case, a client distributes secret shares of its private data to multiple compute nodes, each of which cannot infer the client's data from its own share. Each node trains a network locally on its share of the data, and securely exchanges the computed result with its peers via secure computation primitives like Garbled Circuits [49] and Oblivious Transfer [4, 37]. As it is difficult to generalize such primitives to work with a large number of compute nodes, current MPC-based protocols are limited to two-party [1, 15, 34] and three-party settings [33, 46].

Federated learning (FL), as an emerging distributed learning framework that focuses on participants' data privacy, has each of the distributed participants who has some data locally train a local model with its data, and send the data to a remote server to aggregate into a global model. While the data never leaves the participants, it was shown that much of the information about the data and the identities of the participants themselves can be inferred from the local models received at the server side (see, e.g., [16, 40, 47, 52]). Many techniques based on differential privacy [18, 32, 45] and MPC primitives [10, 22, 41] have been developed to address this issue.

While state-of-the-art HE and MPC based secure deep learning protocols and FL protocols all focus on protecting clients' data privacy, protocols have been recently developed to provide explicit guarantees on both data privacy at remote server and model privacy on the local compute nodes, in the context of *DNN inference* [38]. In this paper, we take one step further to develop a protocol with such *two-sided* privacy requirement for a more challenging problem of *DNN learning*.

The layer-by-layer training approach trains a deep neural network one layer at a time, and was popularized in series of early works on deep learning [8, 26]. Once the model parameters of one layer are trained, they are frozen and used to generate the input data for training the next layer. The layer-by-layer approach has since been superceded by other techniques. However, here we revisit this training architecture due to the privacy constraints. More recent work [25] argued that for training DNNs on the MNIST dataset, the layer-by-layer approach can achieve similar model accuracy with the conventional backpropagation approach in much less time.

II. Secure Distributed DNN Training

A. System and Threat Models

We consider a distributed learning system that consists of M data-owning nodes, N worker nodes, and one Learner node. Each data owner i has a private dataset $(\bar{X}_i, \bar{Y}_i)$ of size m_i, where $\bar{X}_i \in \mathbb{R}^{d \times m_i}$ contains the set of input features of dimension d and $\bar{Y}_i \in \mathbb{R}^{\ell \times m_i}$ contains the corresponding set

of output labels of dimension ℓ. Let $m \triangleq \sum_i m_i$ be the size of the joint dataset (X, Y) created by concatenating $(\bar{X}_i, \bar{Y}_i)$ for all $i \in [M]$. The Learner intends to leverage the computational power of the worker nodes to train a deep feed-forward neural network over the private dataset (X, Y).

We consider a threat model in which the workers are honest-but-curious. In particular, the workers follow the protocol honestly, but may collude amongst themselves to learn additional information about the data and the model. We assume that the Learner is also honest-but-curious, but does not collude with any other parties.

B. Privacy Requirements

We impose *two-sided* privacy requirement on this distributed learning system. Specifically, by the end of the training process, we want the following guarantees:

1) The Learner learns nothing about the private data, except for the weight parameters of the trained DNN.
2) The workers do not learn anything about the trained model nor the private data. This should hold true even if up to T workers collude, for some $T < N$.

C. DNN Architecture

We consider training a feed-forward DNN with $L-1$ hidden layers, and adopt the following approximations to the standard architecture:

1) **Polynomial activation and loss functions:** We use polynomial activation and loss functions instead of the canonical ones such that gradient computation to train each layer can be represented as a multivariate polynomial of the data and the weights.
2) **Layer-by-layer Training**: We train the DNN one layer at a time. Once a layer is trained, its parameters are frozen and used to generate the input data for the next layer. We repeat this process until all L layers are trained.

While these modifications may incur performance loss (e.g., lower test accuracy), we empirically demonstrate that this loss is small. Nevertheless, as we will show later, these approximations enable a scalable design on secure distributed learning that meets the privacy requirements.

Using layer-by-layer training essentially reduces the problem of training a DNN with an arbitrary number of layers to training a shallow 2-layer network. Hence, here we focus on describing the setting for training this shallow network. Later, we will show how to iterate this process to train the entire network.

We consider a shallow neural network with one hidden layer containing h nodes. Let $W \in \mathbb{R}^{h \times d}$ denote the weights from the input layer to the hidden layer, and let $V \in \mathbb{R}^{\ell \times h}$ denote the weights from the hidden layer to the output layer consisting of a single node, where ℓ is the number of labels. A forward pass of the data for a single layer is computed as,

$$Z = W \cdot X, \qquad Z \in \mathbb{R}^{h \times m} \tag{1}$$

$$A = \sigma(Z), \qquad A \in \mathbb{R}^{h \times m} \tag{2}$$

$$\hat{Y} = V \cdot A, \qquad \hat{Y} \in \mathbb{R}^{\ell \times m} \tag{3}$$

where we select $\sigma(\cdot)$ to be the element-wise squaring operation. We make use of polynomial loss functions such as the mean square error given by

$$C(\hat{Y}, Y) = \frac{1}{2m} \sum_{i=1}^{m} ||\hat{Y}_i - Y_i||^2. \tag{4}$$

In this particular case, the gradients of the loss with respect to the weights are given by the following formulas,

$$\frac{\partial C}{\partial V} = \frac{1}{m}(\hat{Y} - Y) \cdot A^\top \tag{5}$$

$$\frac{\partial C}{\partial W} = \frac{2}{m}[V^\top (\hat{Y} - Y) \circ Z] \cdot X^\top \tag{6}$$

where $\circ$ denotes an element-wise multiplication.

In each iteration r of R total iterations, we train the model by performing a standard gradient descent (GD) update,

$$V_r = V_{r-1} - \eta \frac{\partial C}{\partial V_{r-1}} \tag{7}$$

$$W_r = W_{r-1} - \eta \frac{\partial C}{\partial W_{r-1}} \tag{8}$$

where η denotes the learning rate.

III. Building Blocks and Notation

In this section, we lay out the notation used to formally describe the protocol in the next section. This protocol makes use of two main primitives: Shamir Secret Sharing and Lagrange Coded Computing. All operations are performed in a finite field $\mathbb{F}_p$ for some large prime p.

A. Shamir Secret Sharing

We denote a Shamir secret share polynomial of an item X by

$$S[X](u) = X + \sum_{t \in [T]} A_t u^t \tag{9}$$

where the A_t's are uniformly random pads of a finite field and u is the indeterminate. For definiteness, we will affix a superscript when needed to uniquely identify the pads used in a particular encoding, letting A_t^X refer to a pad used to encode X and A_t^W a pad used to encode W. While this notation occasionally becomes somewhat heavy, we aim to include no more identifiers than needed for disambiguation.

Typically, the secret share of X generated for a node j is obtained by evaluating $S[X](u)$ at λ_j, where the λ_j's are a set of unique points known to all nodes. Specifically, we denote $S_j[X] = S[X](\lambda_j)$.

Shamir Secret Sharing is T-private. In other words, it has the property that for any collection $\mathcal{C} \subset [N]$ with $|\mathcal{C}| \leq T$,

$$I(X; (S_j[X])_{j \in \mathcal{C}}) = 0. \tag{10}$$

Conversely, for any collection of $\mathcal{C} \subseteq [N]$, with $|\mathcal{C}| > T$, a node having $(S_j[X])_{j \in \mathcal{C}}$ is able to perfectly reconstruct $S[X](u)$, and hence X.

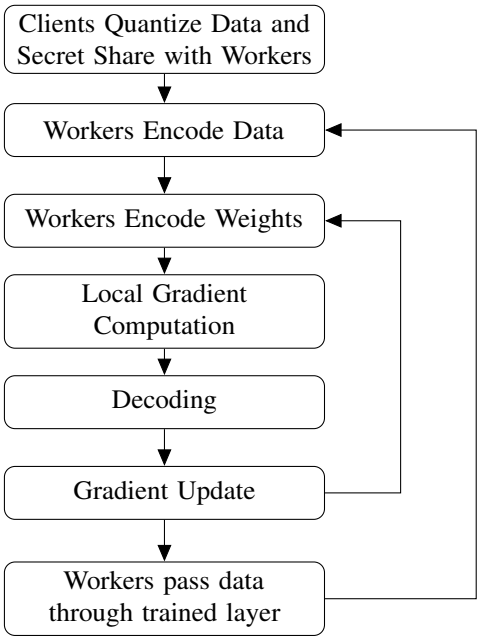

Fig. 2. Task flow of DeepPrivate protocol in training a DNN with arbitrary number of layers.

B. Lagrange Encoding

For a collection[1] of K items $\mathbf{X} = (X_1, \ldots, X_K)$, we denote the Lagrange interpolating polynomial with T padding terms by

$$L[\mathbf{X}](v) = \sum_{k\in[K]} m_k(v)X_k + \sum_{k\in[K+T]\setminus[K]} m_k(v)B_k \quad (11)$$

where the B_k's are uniformly random elements of a finite field, v is the indeterminate, and

$$m_k(v) = \prod_{\ell\in[K+T]\setminus k} \frac{v-\beta_\ell}{\beta_k-\beta_\ell},$$

where the β_k's are a set of unique points. As previously, for a collection of unique α_k's (each distinct from any β_k), we let

$$L_j[\mathbf{X}] = L[\mathbf{X}](\alpha_j). \quad (12)$$

Like Shamir Secret Sharing, Lagrange encodings are T-private. Since $L[\mathbf{X}](v)$ is a $(K+T-1)$-degree polynomial in v, for any collection of $\mathcal{C} \subseteq [N]$, with $|\mathcal{C}| \geq K+T$, a node having $(L_j[\mathbf{X}])_{j\in\mathcal{C}}$ is able to perfectly reconstruct $L[\mathbf{X}](v)$, and hence $\mathbf{X}$. For definiteness, we use notations such as $B_k^{\mathbf{X}}$ to denote the pads used to encode $\mathbf{X}$.

IV. DeepPrivate Protocol

In this section we focus on describing the proposed DeepPrivate protocol for training a single layer of a DNN, and how we obtain the data for training the next layer, which repeats the same training process.

[1]In general, boldface variables will denote collections of K items.

A. Quantization and Secret Sharing

Each data node (data owner) quantizes its local data into a finite field $\mathbb{F}_p$ for some large p, and distributes the quantized data to the workers via Shamir secret shares. We leave the details of quantization to the supplementary materials. For convenience and clarity, we use $\bar{X}_i$ to denote the quantized dataset of data node i (this includes both the input features and the output labels). Each data node i creates and sends a secret share $S_j[\bar{X}_i]$ of $\bar{X}_i$ to each worker node $j \in [N]$. Once worker j receives $S_j[\bar{X}_i]$ from each $i \in [M]$, it concatenates them to form its secret share of the joint dataset, denoted by $S_j[X]$ where $X = [\bar{X}_1, \ldots, \bar{X}_M]$. With some design parameter $K < N$, each worker j partitions this secret share $S_j[X]$ into K parts of equal size, which we denote collectively by $\mathbf{S}_j[\mathbf{X}] \triangleq (S_j[X_k])_{k\in[K]}$ for the partition $\mathbf{X} \triangleq [X_1, \ldots, X_K]$.

B. Dataset and Weight Encoding

To achieve computational efficiency, our protocol has each worker process $1/K$ of the entire dataset when computing gradients, with Lagrange Coded Computing to protect data and model privacy. To do this, before the training starts, workers encode their local secret shares and communicate with each other for each worker i to obtain the Lagrange encoding $L_i[\mathbf{X}]$.

Before proceeding to describe the encoding procedures for data and weights, we note that the proposed DeepPrivate employs Commodity-Based Cryptography [5], and relies on a Trusted Initializer (TI) to provide random pads that do not depend on the inputs (i.e., the training data and the weights) to the protocol. This type of functionality is used in other secure MPC protocols [44]. Importantly, a TI is distinct from a Trusted Intermediary, since the TI has no access to any private data.

Before the training of the current layer starts, the TI samples T random pads $(B_k^{\mathbf{X}})_{k\in[K+T]\setminus[K]}$ uniformly at random, and sends their secret shares evaluated at λ_j, i.e., $(S_j[B_k^{\mathbf{X}}])_{k\in[K+T]\setminus[K]}$ to worker j (note that $S_j[B_k^{\mathbf{X}}]$ itself is constructed using additional pads denoted by $(A_t^{B_k})_{t\in[T]}$). Using these secret shares as pads, worker j constructs the following polynomial

$$L[\mathbf{S}_j[\mathbf{X}]](v) = \sum_{k\in[K]} m_k(v)S_j[X_k] + \sum_{k\in[K+T]\setminus[K]} m_k(v)S_j[B_k^{\mathbf{X}}], \quad (13)$$

evaluates the polynomial α_i to obtain $L_i[\mathbf{S}_j[\mathbf{X}]]$, and sends it to worker i, for all $i \in [N]$.

It is straightforward to show (as we do in the supplementary material) that under this procedure, once worker i receives $L_i[\mathbf{S}_j[\mathbf{X}]]$ from any subset of $T+1$ workers, it can use them as valid interpolation points to recover $L_i[\mathbf{X}]$.

Before the training of the current layer starts, each worker j is given secret shares of the initial weights W_0 and V_0, denoted by $S_j[W_0]$ and $S_j[V_0]$ respectively. In each iteration

$r = 1, \ldots, R$ of the training process, each worker j performs Lagrange encoding on $S_j[W_{r-1}]$ and $S_j[V_{r-1}]$ respectively as

$$L[S_j[W_{r-1}]](v) = \sum_{k\in[K]} m_k(v)S_j[W_{r-1}] + \sum_{k\in[K+T]\setminus[K]} m_k(v)S_j[B_k^{W_{r-1}}]; \quad (14)$$

$$L[S_j[V_{r-1}]](v) = \sum_{k\in[K]} m_k(v)S_j[V_{r-1}] + \sum_{k\in[K+T]\setminus[K]} m_k(v)S_j[B_k^{V_{r-1}}], \quad (15)$$

with the random pads $(S_j[B_k^{W_{r-1}}], S_j[B_k^{V_{r-1}}])_{k\in[K+T]\setminus[K]}$ received from the TI. Then worker j evaluates these two polynomials at α_i and sends the results $L_i[S_j[W_{r-1}]]$ and $L_i[S_j[W_{r-1}]]$ to worker i, for all $i \in [N]$. Similarly as for the data, worker i recovers $L_i[W_{r-1}]$ and $L_i[V_{r-1}]$ after receiving messages from any subset of $T+1$ workers. To summarize, at the beginning of iteration r of training a certain layer, each worker i has $L_i[\mathbf{X}], L_i[W_{r-1}], L_i[V_{r-1}]$.

C. Local Computations and Secret Sharing of Coded Results

In iteration r of the training process, the workers perform gradient computation in the LCC domain. That is, letting

$$f_V(\mathbf{X}, V, W) = m\frac{\partial C}{\partial V}, \quad (16)$$

$$f_W(\mathbf{X}, V, W) = m\frac{\partial C}{\partial W}, \quad (17)$$

each node i computes

$$f_V(L_i[\mathbf{X}], L_i[W_{r-1}], L_i[V_{r-1}]) = q_V(\alpha_i), \quad (18)$$

$$f_W(L_i[\mathbf{X}], L_i[W_{r-1}], L_i[V_{r-1}]) = q_W(\alpha_i), \quad (19)$$

where $q_V(v)$ and $q_W(v)$ are polynomials of degrees $\deg(f_V)(K+T-1)$ and $\deg(f_W)(K+T-1)$ respectively. By carefully examining the gradient expressions in (5) and (6), we have $\deg(f_V) = \deg(f_W) = 9$.

Having computed the gradients on their local Lagrange encoded data, workers create secret shares of computation results and communicate with each other, such that each worker can decode a secret share of the gradient computed over the entire dataset $\mathbf{X}$. As such operations are identical for the weights W and V, we describe the operation once in the following without specifying the subscript for the function f.

For each $t = 1, \ldots, T$, the TI samples K random pads $A_t^{G_1}, , \ldots, A_t^{G_K}$, one for each $G_k \triangleq f(X_k, W, V)$. Next, for each t, the TI performs Lagrange encoding on the bundle $\mathbf{A}_t^{\mathbf{G}} \triangleq (A_t^{G_k})_{k\in[K]}$, and distributes the T coded random pads $(L_j[\mathbf{A}_t^{\mathbf{G}}])_{t\in[T]}$ to each worker $j \in [N]$. Note that to maintain security, we have the degrees of the Lagrange polynomials created at the TI match the degree of $q(v)$. We provide the explicit constructions of the Lagrange encodings in the supplementary material.

Having received the coded random pads $(L_j[\mathbf{A}_t^{\mathbf{G}}])_{t\in[T]}$ from the TI, each worker j uses them as pads to secret share its local computation result $q(\alpha_j)$. Specifically, worker j sends $S_i[q(\alpha_j)]$ to each worker $i \in [N]$ where

$$S_i[q(\alpha_j)] = q(\alpha_j) + \sum_{t\in[T]} L_j[\mathbf{A}_t^{\mathbf{G}}]\lambda_i^t \quad (20)$$

D. Decoding

Worker i waits to collect $P = 9(K+T-1)+1$ secret shares $(S_i[q(\alpha_j)])_{j\in\mathcal{I}_i}$ where $\mathcal{I}_i$ denotes the set of P workers whose messages are received at worker i first. In the supplementary material, we show that by interpolating these secret shares at points $(\alpha_j)_{j\in\mathcal{I}_i}$, worker i obtains a polynomial $p_i(v)$ of degree $P-1$. Next, worker i evaluates $p_i(v)$ at the points $\beta_1, \ldots, \beta_K$ such that

$$p_i(\beta_k) = G_k + \sum_{t\in[T]} A_t^{G_k}\lambda_i^t = S_i[G_k]. \quad (21)$$

These secret shares $S_i[G_1], \ldots, S_i[G_K]$ are then aggregated to form a secret share of the full gradient over the entire dataset $\mathbf{X}$ at worker i:

$$S_i[f(\mathbf{X}, W, V)] = \sum_{k\in[K]} S_i[G_k]. \quad (22)$$

E. Model Update

In this stage, the workers multiply their secret shares of the gradient by the learning rate and subtract the product from the current weights. Specifically, in iteration r, the workers undergo a Secure Truncation protocol [12] to obtain secret shares of $\frac{\eta}{m}f_V(\mathbf{X}, V_r, W_r)$ and $\frac{\eta}{m}f_W(\mathbf{X}, V_r, W_r)$. Then each worker i recovers a secret share of the new weights as follows:

$$S_i[V_{r+1}] = S_i[V_r] - S_i\left[\frac{\eta}{m}f_V(\mathbf{X}, V_r, W_r)\right], \quad (23)$$

$$S_i[W_{r+1}] = S_i[W_r] - S_i\left[\frac{\eta}{m}f_W(\mathbf{X}, V_r, W_r)\right]. \quad (24)$$

It is important to note that we cannot retain full precision when multiplying the gradient by $\frac{\eta}{m} < 1$. If we wanted to do so, the size of $\mathbb{F}_p$ would have to increase exponentially with the number of iterations. Thus, the Secure Truncation protocol we use introduces some noise. However, note that since the protocol produces an unbiased estimator of the true gradient, the training process still converges in spite of this introduced noise.

F. Sending Results to the Learner

During the training of one layer of the neural network, once the workers have performed R iterations, and each of them has a secret share of the final weights W_R, the workers must send these secret shares to the Learner so that the Learner can recover the weights in the clear.

Each worker i proceeds to send $S_i[W_R]$ to the Learner. The Learner can then recover the final weights W_R whenever it receives $T+1$ shares.

G. Moving to the Next Layer

By the end of training a particular layer with the trained weights W_R, each worker i has a secret share of the input data $S_i[X]$ and a secret share of the weights $S_i[W_R]$. The workers undergo a T-private secure MPC protocol (e.g., the BGW protocol) to compute a secret share $S_i[\sigma(W_R \cdot X)]$ at each worker i. Note that we will need to perform degree reduction twice to maintain a degree-T secret share of $\sigma(W_R \cdot X)$ at each worker. Treating these secret shares as their inputs to the next layer, as shown in Figure 2, the workers repeat the above steps to train the next layer.

V. Correctness and Privacy Guarantees

We begin by formally stating the recovery threshold for the correctness our protocol, which stems from the requirement that the workers interpolate a polynomial of degree $9(K+T-1)$ during each iteration. We present a formal proof of the theorem in the supplementary materials.

Theorem 1. *For any $N \geq 9(K+T-1)+1$, the proposed deepPrivate protocol correctly performs gradient descent such that for each iteration r,*

$$V_{r+1} = V_r - \eta \frac{\partial C}{\partial V_r} + n_r^V, \tag{25}$$

$$W_{r+1} = W_r - \eta \frac{\partial C}{\partial W_r} + n_r^W, \tag{26}$$

for some additive noises n_r^V and n_r^W with $\mathbb{E}[n_r^V] = \mathbb{E}[n_r^W] = 0$, and $\mathbb{E}[||n_r^V||_2^2] \leq \sigma^2$, $\mathbb{E}[||n_r^W||_2^2] \leq \sigma^2$ for some σ^2 defined by the Secure Truncation Protocol.

It is well known that gradient descent is robust to the additive noise mentioned in the theorem statement; indeed, additive noise has long been used during training as a regularization technique to aid in generalization [9].

We next state the privacy guarantees of our protocol, amounting to data privacy on the learner side, and data and model privacy on the client side.

A. Learner Side Privacy

We start by demonstrating privacy on the learner side.

Theorem 2. *The only thing the Learner learns are the final weights of each layer. In particular, for each layer,*

$$\mathbf{P}(X|(S_j[W_R])_{j\in[N]})) = \mathbf{P}(X|W_R) \tag{27}$$

Proof. Since each are $T+1$ evaluation points of the same T-degree polynomial, there exists a bijection between $(S_j[W_R])_{j\in[N]}$ and $(W_R, (A_t)_{t\in[T]})$. Here, $(A_t)_{t\in[T]}$ denotes the collection of random pads that are masking W_R. Thus, we have

$$\mathbf{P}(X|(S_j[W_R])_{j\in[N]}) = \mathbf{P}(X|W_R, (A_t)_{t\in T}) \tag{28}$$

$$= \mathbf{P}(X|W_R), \tag{29}$$

with the final equality because the pads are independent of X and W_R. □

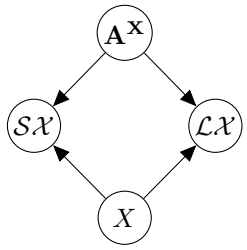

Fig. 3. Dependencies between $\mathcal{SX}$ and $\mathcal{LX}$, where $\mathbf{A^X}$ is shorthand for $(\mathbf{A}_t^{\mathbf{X}})_{t\in[T]}$

B. Client Side Privacy

Our client security theorem guarantees that for any subset $\mathcal{C}$ of up to T colluding workers, they cannot learn anything about W, V, or X from the set of messages received by those workers.

Theorem 3. *For any subset $\mathcal{C}$ of up to T colluding workers, we have*

$$\mathbf{P}(W_R|((\mathcal{M}_{i,r})_{r\in[R]})_{i\in\mathcal{C}}) = \mathbf{P}(W_R), \tag{30}$$

$$\mathbf{P}(X|((\mathcal{M}_{i,r})_{r\in[R]})_{i\in\mathcal{C}}) = \mathbf{P}(X), \tag{31}$$

where $\mathcal{M}_{i,r,l}$ represents the messages received at worker i in iteration r.

This theorem declares client side privacy for a single layer of training. However, it is clear to see how privacy extends to the whole protocol. Note that the "output" of each layer from the clients' perspective is a secret share of the weights W_R^ℓ. As we show in the proof of the theorem, this secret share (and its pads) is independent of any messages sent between the clients during the training of layer ℓ. Thus, no messages sent during a previous layer hold any information about messages sent in subsequent ones.

In fact, we rely on this independence between messages and secret shares to prove privacy within a layer as well. A preliminary step in proving the theorem is to group the messages $\mathcal{M}_{i,r}$ into "non-intersecting" groups, in the sense that no two groups depend on the same random pads. However, this is only possible up to a point, due to the interaction between secret sharing and lagrange encoding present in the protocol. This interaction is reflected in the existence of collections messages such as $\mathcal{SX} \triangleq (S_i[\mathbf{X}])_{i\in\mathcal{C}}$ and $\mathcal{LX} \triangleq ((L_i[\mathbf{S}_j[\mathbf{X}]])_{j\in[N]})_{i\in\mathcal{C}}$ both of which are constructed using $\mathbf{A^X} \triangleq (\mathbf{A}_t^{\mathbf{X}})_{t\in[T]}$ (See Figure 3). To establish independence between these groups, we first make use of two "commutation lemmas" illustrate that the protocols described in Sections IV-B and IV-D are secure. . The lemmas allow us to "separate" the messages sent into pure secret shares or pure lagrange shares of either values of interest or pads, and in turn simply demonstrate the desired independence.

VI. Complexity Analysis

In this section, we compare the complexity of the proposed DeepPrivate with BGW protocol [7], a celebrated secure MPC

TABLE I

COMPARISON OF COMPLEXITY PER WORKER OF PROPOSED DEEPPRIVATE PROTOCOL AND BGW PROTOCOL FOR TRAINING ONE LAYER OF A DNN. HERE, N IS THE NUMBER OF WORKERS, m IS THE TOTAL NUMBER OF DATA POINTS, d IS THE INPUT DIMENSION, ℓ IS THE OUTPUT DIMENSION, h IS THE NUMBER OF NODES IN THE HIDDEN LAYER, AND R IS THE NUMBER OF TRAINING ITERATIONS.

Complexity	**DeepPrivate**	**BGW**
Gradient computation	$O(\frac{mh(d+\ell)}{K}R)$	$O\left(mh(d+\ell)R\right)$
LCC coding	$O\left(\frac{mdN(\log N)}{K}+h(d+\ell)N(\log N)R\right)$	None
Communication between workers	$O\left(\frac{md}{K}N+h(d+\ell)NR\right)$	$O\left(m(h+\ell)R+h(d+\ell)R\right)$ broadcasts
Required no. of workers	$N\geq 9(K+T-1)+1$	$N\geq 2T+1$

protocol, for training a single DNN layer. Specifically, using BGW, each worker directly computes (5) and (6) on its secret shares of the data and the weights to obtain a secret share of the gradient. Also, we consider the implementation of BGW with degree reduction (optimized version as in [6]), where the degree of secret polynomial is maintained at T after each multiplication of secret shares.

We provide the detailed analysis of the complexities of the two protocols in the supplementary material, and summarize the results in Table I. We can see that the proposed DeepPrivate protocol improves the complexity of the expensive gradient computation (especially for big datasets with large m) by $K\times$ over the BGW protocol. Specifically, DeepPrivate workers perform gradient computations over Lagrange coded data partitions with size that is $\frac{1}{K}$ of the entire dataset. This demonstrates the scalability of the DeepPrivate protocol, i.e., when more workers join the system, we can have the number of data partitions $K=\Theta(N)$ scale linearly with N. Then, we can either process more data by setting $m=\Theta(N)$ while keeping the computational load of each worker constant, or fix m and have each worker process less amount of data, hence speeding up the training process. Also, we observe that DeepPrivate can protect data and model privacy from a constant fraction of the workers no matter how large the network size N is, e.g., by setting $T=\Theta(N)$.

VII. EMPIRICAL RESULTS

In this section, we provide experimental results on training a seven-layer neural network LeNet-5 [28] using DeepPrivate on three different datasets: MNIST [27], Fashion-MNIST [48] and SVHN [35]. Specifically, we adopt DeepPrivate with the layer-by-layer training architecture, MSE loss function, and quadratic activation function.

For comparison, we also train LeNet-5 with the gold standard end-to-end training with backpropogation, cross-entropy loss, and ReLU activation function; and a linear model using logistic regression with the COPML protocol.[2]

We can see in Figure 4 that for all datasets, LeNet-5 trained by the gold standard approach and DeepPrivate consistently outperforms the linear model trained by COPML by a large margin, demonstrating the effectiveness of the proposed approximations in DeepPrivate in picking up the power of deep neural networks.

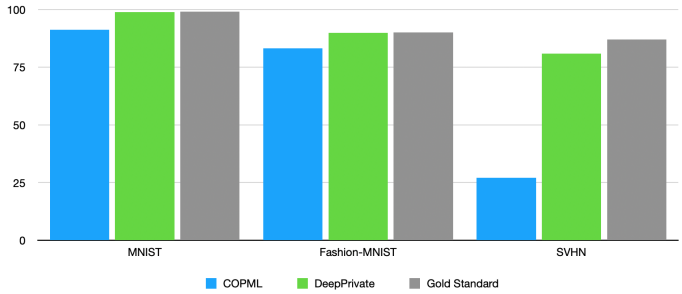

Fig. 4. Test accuracy comparison of training a DNN LeNet-5 and a linear model on different datasets.

Moreover, DeepPrivate incurs no loss on the relatively simple MNIST and Fashion-MNIST datasets, and a marginal hit on SVHM, compared with the gold standard training algorithm. We suspect the loss of performance attributes to 1) lack of flexibility of updating earlier layers during layer-by-layer training (as they are frozen after being trained); and 2) numerical instability caused by polynomial activation and loss functions. Optimizing DeepPrivate to minimize performance loss on more complicated datasets remains as an important future research.

VIII. CONCLUSION AND FUTURE RESEARCH

DeepPrivate uses state-of-the-art coding techniques and layer-by-layer training to achieve distributed training of a neural network in an information-theoretically private manner. Future research on using layer-by-layer training and polynomial activation functions for more complicated datasets would allow for DeepPrivate to be used for a wider range of problems.

[2]As the polynomial approximation of the logistic function in COPML was designed for binary classification, and does not naturally generalize to the multi-class scenario we are considering here, we keep the cross-entropy loss for COPML and expect the observed test accuracy as an upper bound on what would have been achieved by COPML.

References

[1] Agrawal, N., Shahin Shamsabadi, A., Kusner, M. J., and Gascón, A. QUOTIENT: two-party secure neural network training and prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1231–1247, 2019.

[2] Aono, Y., Hayashi, T., Trieu Phong, L., and Wang, L. Scalable and secure logistic regression via homomorphic encryption. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pp. 142–144, 2016.

[3] Aono, Y., Hayashi, T., Wang, L., Moriai, S., et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2017.

[4] Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 535–548, 2013.

[5] Beaver, D. Commodity-based cryptography (extended abstract). In *STOC '97*, 1997.

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

[7] Ben-Or, M., Goldwasser, S., and Wigderson, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pp. 351–371. 2019.

[8] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[9] Bishop, C. M. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

[10] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

[11] Bost, R., Popa, R. A., Tu, S., and Goldwasser, S. Machine learning classification over encrypted data. In *NDSS*, volume 4324, pp. 4325, 2015.

[12] Catrina, O. and Saxena, A. Secure computation with fixed-point numbers. In Sion, R. (ed.), *Financial Cryptography and Data Security*, pp. 35–50, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-14577-3.

[13] Chen, V., Pastro, V., and Raykova, M. Secure computation for machine learning with spdz. *arXiv preprint arXiv:1901.00329*, 2019.

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

[15] Demmler, D., Schneider, T., and Zohner, M. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[16] Geiping, J., Bauermeister, H., Dröge, H., and Moeller, M. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020.

[17] Gentry, C. et al. *A fully homomorphic encryption scheme*, volume 20. Stanford university Stanford, 2009.

[18] Geyer, R. C., Klein, T., and Nabi, M. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[19] Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pp. 201–210. PMLR, 2016.

[20] Graepel, T., Lauter, K., and Naehrig, M. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pp. 1–21. Springer, 2012.

[21] Han, K., Hong, S., Cheon, J. H., and Park, D. Logistic regression on homomorphic encrypted data at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 9466–9471, 2019.

[22] He, L., Karimireddy, S. P., and Jaggi, M. Secure byzantine-robust machine learning. *arXiv preprint arXiv:2006.04747*, 2020.

[23] Hesamifard, E., Takabi, H., and Ghasemi, M. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[24] Hesamifard, E., Takabi, H., Ghasemi, M., and Wright, R. N. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies*, 2018 (3):123–142, 2018.

[25] Hettinger, C., Christensen, T., Ehlert, B., Humpherys, J., Jarvis, T., and Wade, S. Forward thinking: Building and training neural networks one layer at a time. *arXiv preprint arXiv:1706.02480*, 2017.

[26] Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, 18 (7):1527–1554, 2006.

[27] LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[28] LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pp. 53–60. Perth, Australia, 1995.

[29] Li, P., Li, J., Huang, Z., Gao, C.-Z., Chen, W.-B., and Chen, K. Privacy-preserving outsourced classification in cloud computing. *Cluster Computing*, 21(1):277–286, 2018.

[30] Lindell, Y. and Pinkas, B. Privacy preserving data mining.

In *Annual International Cryptology Conference*, pp. 36–54. Springer, 2000.

[31] McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282. PMLR, 2017.

[32] McMahan, H. B., Ramage, D., Talwar, K., and Zhang, L. Learning differentially private recurrent language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=BJ0hF1Z0b.

[33] Mohassel, P. and Rindal, P. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 35–52, 2018.

[34] Mohassel, P. and Zhang, Y. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 19–38. IEEE, 2017.

[35] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. 2011.

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

[37] Peikert, C., Vaikuntanathan, V., and Waters, B. A framework for efficient and composable oblivious transfer. In *Annual international cryptology conference*, pp. 554–571. Springer, 2008.

[38] Rouhani, B. D., Riazi, M. S., and Koushanfar, F. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

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

[40] Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18. IEEE, 2017.

[41] So, J., Güler, B., and Avestimehr, A. S. Byzantine-resilient secure federated learning. *IEEE Journal on Selected Areas in Communications*, 2020.

[42] So, J., Guler, B., and Avestimehr, A. S. A scalable approach for privacy-preserving collaborative machine learning. *arXiv preprint arXiv:2011.01963*, 2020.

[43] Sun, X., Zhang, P., Liu, J. K., Yu, J., and Xie, W. Private machine learning classification based on fully homomorphic encryption. *IEEE Transactions on Emerging Topics in Computing*, 8(2):352–364, 2018.

[44] Tonicelli, R., Nascimento, A. C. A., Dowsley, R., Müller-Quade, J., Imai, H., Hanaoka, G., and Otsuka, A. Information-theoretically secure oblivious polynomial evaluation in the commodity-based model. *International Journal of Information Security*, pp. 73–84, 2014.

[45] Truex, S., Liu, L., Chow, K.-H., Gursoy, M. E., and Wei, W. Ldp-fed: Federated learning with local differential privacy. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, pp. 61–66, 2020.

[46] Wagh, S., Gupta, D., and Chandran, N. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019 (3):26–49, 2019.

[47] Wang, Z., Song, M., Zhang, Z., Song, Y., Wang, Q., and Qi, H. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2512–2520. IEEE, 2019.

[48] Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[49] Yao, A. C. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pp. 160–164. IEEE, 1982.

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

[51] Yuan, J. and Yu, S. Privacy preserving back-propagation neural network learning made practical with cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):212–221, 2013.

[52] Zhu, L. and Han, S. Deep leakage from gradients. In *Federated Learning*, pp. 17–31. Springer, 2020.

APPENDIX A
CORRECTNESS PROOF

In this section, we state and prove two lemmas that will help with the proofs of both Theorems 1 and 3. We then prove Theorem 1.

In this and the following section, for the clarity of exposition, we state and prove results with respect to X and W only. However, the arguments are fully generic and extend to the full case in which calculations involve the full set of variables X, W, and V without modification.

Before proceeding to the important lemmas and proofs, we specify the form of the modified Lagrange encodings, $(L_i[\mathbf{A}_t^{\mathbf{G}}])_{t\in[T]}$ generated and distributed by the Trusted Initializer in Section IV-D. To maintain security, the Trusted Initializer must match the degree of the encoding Lagrange polynomial to that of $q(v)$, which equals to $P-1 = 9(K+T-1)$. Thus, each Lagrange encoding looks like

$$L_i[\mathbf{A}_t^{\mathbf{G}}] = \sum_{k\in[K]} m_k(\alpha_i)A_t^{G_k} + \sum_{k\in[P]\setminus[K]} m_k(\alpha_i)B_k^{\mathbf{A}_t}, \quad (32)$$

where the $B_k^{\mathbf{A}_t}$'s are pads known only to the Trusted Initializer and we have implicitly expanded the β_k's to be a collection of P unique points.

We now state and prove the commutation lemmas.

Lemma 4. *For a bundle of items $\mathbf{X}$ and any collection of nodes $\mathcal{C}$ with $|\mathcal{C}| \geq T+1$, there is a bijection (with the λ_j's as common knowledge):*

$$(L_i[\mathbf{S}_j[\mathbf{X}]])_{j\in\mathcal{C}} \longleftrightarrow (L_i[\mathbf{X}], (L_i[\mathbf{A}_t^{\mathbf{X}}])_{t\in[T])}). \quad (33)$$

where

$$L_i[\mathbf{A}_t^{\mathbf{X}}] = \sum_{k\in[T]} m_k(\alpha_i)A_t^{X_k} + \sum_{k\in[K+T]\setminus[K]} m_k(\alpha_i)A_t^{B_k}$$

Proof. Simple algebraic manipulation shows that

$$L_i[S_j[\mathbf{X}]] = L_i[\mathbf{X}] + \sum_{t\in[T]}\left(\sum_{k=1}^{K} m_k(\alpha_i)A_t^{X_k} + \sum_{k=K+1}^{K+T} m_k(\alpha_i)A_t^{B_k}\right)\lambda_j^t \quad (34)$$

$$= L_i[\mathbf{X}] + \sum_{t\in[T]} L_i[\mathbf{A}_t^{\mathbf{X}}]\lambda_j^t \quad (35)$$

which corresponds to a degree-T polynomial evaluated at λ_j (note that the only dependence on j is through λ_j).

Given a collection $(L_i[S_j[\mathbf{X}]])_{j\in\mathcal{C}}$, by interpolating the polynomial, we can uniquely determine $L_i[\mathbf{X}]$ and $(L_i[\mathbf{A}_t^{\mathbf{X}}])_{t\in[T]}$. Conversely, given these quantities, we can reconstruct the polynomial and evaluate them at $(\lambda_j)_{j\in\mathcal{C}}$ to uniquely obtain $(L_i[S_j[\mathbf{X}]])_{j\in\mathcal{C}}$. □

Lemma 5. *At each worker i, for any subset of nodes $\mathcal{C}$ with $|\mathcal{C}| \geq P$ and a collection of corresponding computation results $(f(L_j[\mathbf{X}], L_j[W]))_{j\in\mathcal{C}}$, there is a bijection (with the α_j's and β_k's as common knowledge):*

$$(S_i[f(L_j[\mathbf{X}], L_j[W])])_{j\in\mathcal{C}} \longleftrightarrow ((S_i[f(X_k, W)])_{k\in[K]}, (S_i[\xi_k])_{k\in[P]\setminus[K]}), \quad (36)$$

where

$$S_i[\xi_k] = \xi_k + \sum_{t\in[T]} B_k^{\mathbf{A}_t}\lambda_i^t$$

and $\xi_k := f(B_k^{\mathbf{X}}, B_k^W)$ for $k \in [K+T]\setminus[K]$ and $\xi_k = f(L[\mathbf{X}](\beta_k), L[W](\beta_k))$ for $k \in [P]\setminus[K+T]$

Proof. Each worker i receives evaluation points of the form

$$p_i(\alpha_j) = S_i[q(\alpha_j)] = q(\alpha_j) + \sum_{t\in[T]} L_j[\mathbf{A}_t^{\mathbf{G}}]\lambda_i^t, \quad (37)$$

for each $j \in \mathcal{C}$. Recall that the $q(\alpha_j)$'s correspond to evaluations of a degree-$(P-1)$ polynomial $q(v) = f(L[\mathbf{X}](v), L[V](v), L[W](v))$. The second term, as shown in (32), corresponds to the evaluation of a degree-$(P-1)$ polynomial $\tilde{q}_i(v)$ at α_j, where

$$\tilde{q}_i(v) = \sum_{t\in[T]}\sum_{k\in[K]} m_k(v)A_t^{G_k}\lambda_i^t + \sum_{t\in[T]}\sum_{k\in[P]\setminus[K]} m_k(v)B_k^{\mathbf{A}_t}\lambda_i^t. \quad (38)$$

Now let us consider the evaluations of the polynomial $p_i(v) = q(v) + \tilde{q}_i(v)$ at the P unique points defined by the β_k's: for $k \in [K]$, we have

$$p_i(\beta_k) = f(X_k, W) + \sum_{t\in[T]} A_t^{G_k}\lambda_i^t, \quad (39)$$

for $k \in [K+T]\setminus[K]$, we have

$$p_i(\beta_k) = f(B_k^{\mathbf{X}}, B_k^W) + \sum_{t\in[T]} B_k^{\mathbf{A}_t}\lambda_i^t, \quad (40)$$

and for $k \in [P]\setminus[K+T]$, we have

$$p_i(\beta_k) = f(L[\mathbf{X}](\beta_k), L[W](\beta_k)) + \sum_{t\in[T]} B_k^{\mathbf{A}_t}\lambda_i^t \quad (41)$$

As we have shown that both the LHS and the RHS of the lemma statement are a collection of P distinct points on the polynomial $p_i(v)$, the bijection follows. □

Finally, we proceed to prove Theorem 1. Technically, we show that the learner receives a valid secret share of W_R, after running R iterations of the proposed DeepPrivate protocol.

Proof of Theorem 1. To start, we fix $N \geq 9(K+T-1)+1$.

We first note that via initialization, each worker i is supplied with a secret share $S_i[W_0]$. Then, via distribution from the data nodes, each worker i obtains a secret share of $S_i[X_k]$ for each $k \in [K]$. By receiving $L_i[\mathbf{S}_j[\mathbf{X}]]$ from each j in any group of $T+1$ workers, Lemma 4 guarantees that node i can reconstruct $L_i[\mathbf{X}]$.

We now show the main result through an inductive argument. Assuming that at the beginning of iteration r, each node i has valid secret share, $S_i[W_r]$ and Lagrange encoding $L_i[\mathbf{X}]$, it suffices to show that at the end the iteration, each node i has a valid secret share $S_i[W_{r+1}]$ where

$$W_{r+1} = W_r - \eta\frac{\partial C}{\partial W} + n. \quad (42)$$

We review the steps needed for this calculation, along with the recovery thresholds for each step:

1) By receiving $L_i[\mathbf{S}_j[W_r]]$ from each j in any group of $T+1$ workers, Lemma 4 guarantees that node i can reconstruct $L_i[W_r]$.
2) Given Lagrange encodings $L_i[\mathbf{X}]$ and $L_i[W_r]$, node i can locally calculate the gradient $f(L_j[\mathbf{X}], L_j[W_r])$.
3) Next, by receiving $S_i[f(L_j[\mathbf{X}], L_j[W_r])]$ from each j in any group of P workers, Lemma 5 guarantees that node i can recover $S_i[f(X_k, W_r)]$ for all $k \in [K]$. By summing over k, worker i obtains $S_i[m\frac{\partial C}{\partial W_r}]$.
4) The Secure Truncation Protocol guarantees that starting with secret shares of the form $S_i[m\frac{\partial C}{\partial W_r}]$, a collection of $N \geq 2T+1$ nodes can distribute secret shares of $\eta\frac{\partial C}{\partial W_r} + n$, where n satisfies $\mathbb{E}[n] = 0$, $\mathbb{E}[||n||_2^2] \leq \sigma^2$ for some σ^2 defined by the Secure Truncation Protocol [42].
5) Finally, by subtracting the secret share of $\eta\frac{\partial C}{\partial W_r} + n$ from the initial the secret share of W_r, node i obtains the result stated in (42).

The theorem follows as all above steps can be performed successfully when supplied with $N \geq 9(K+T-1)+1$ computation results. □

Appendix B
Client-Side Privacy Proof

In this section, we prove Theorem 3. Before carrying out the proof, we briefly review the messages received by the nodes in a subset $\mathcal{C}$ over the course of the execution of the protocol, and restate the theorem based on some groupings of the messages.

At the outset, the nodes receive secret shares of both the initial weights and data, which we denote as follows:

$$\mathcal{SX} = (S_i[\mathbf{X}])_{i\in\mathcal{C}}$$
$$\mathcal{SW}_0 = (S_i[W_0])_{i\in\mathcal{C}}.$$

The secret shares of $\mathbf{X}$ are redistributed as Lagrange encodings for the sake of Lagrange coded computing:

$$\mathcal{LX} = ((L_i[\mathbf{S}_j[\mathbf{X}]])_{j\in[N]}, (S_i[B_k^{\mathbf{X}}])_{k\in[T]})_{i\in\mathcal{C}}$$

In this case the nodes also know secret shares of Lagrange pads, which we include within the message bundle.

In each iteration $r \in [R]$, secret shares of W_r are also redistributed as Lagrange encodings to enable Lagrange coded computing:

$$\mathcal{LW}_r = ((L_i[\mathbf{S}_j[W_r]])_{j\in[N]}, (S_i[B_k^{W_r}])_{k\in[T]})_{i\in\mathcal{C}}$$

Gradient computations are applied on the Lagrange encoded data and weights, and the coded results are secret shared:

$$\mathcal{G}_r = ((S_i[f(L_j[\mathbf{X}], L_j[W_r])])_{j\in[N]}, (L_i[\mathbf{A}_t^{\mathbf{G},r}])_{t\in[T]})_{i\in\mathcal{C}}$$

In this case, the nodes also know Lagrange encodings of secret share pads, which we include within the bundle.

Finally, a secure truncation protocol is used to perform the gradient update, resulting in secret shares of the new weights.

$$\mathcal{SW}_r = (\mathcal{M}_{i,r}^{STP})_{i\in\mathcal{C}} = (S_i[W_r])_{i\in\mathcal{C}} \tag{43}$$

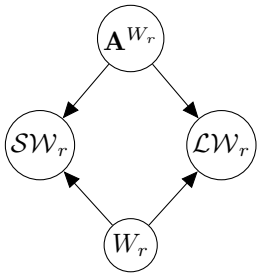

Fig. 5. Dependencies between $\mathcal{SW}_r$ and $\mathcal{LW}_r$, where $\mathbf{A}^{W_r}$ is shorthand for $(A_t^{W_r})_{t\in[T]}$.

Since the secure truncation protocol is known to be T-private, we only consider the resulting secret share in our security analysis.

Next, we restate Theorem 3 with the above defined groups of messages in the following Theorem 6. We note that proving Theorem 6 suffices to show client-side privacy.

Theorem 6. *For any subset $\mathcal{C}$ of workers with $|\mathcal{C}| \leq T$, we have*

$$\mathbf{P}(W_R|\mathcal{SX}, \mathcal{LX}, (\mathcal{SW}_r, \mathcal{LW}_r, \mathcal{G}_r)_{r\in[R]}) = \mathbf{P}(W_R), \tag{44}$$
$$\mathbf{P}(X|\mathcal{SX}, \mathcal{LX}, (\mathcal{SW}_r, \mathcal{LW}_r, \mathcal{G}_r)_{r\in[R]}) = \mathbf{P}(X). \tag{45}$$

Proof. By applying the commutation lemmas, we can simplify some of the groupings

$$\mathcal{LX} = (L_i[\mathbf{X}], (L_i[\mathbf{A}_t^{\mathbf{X}}])_{t\in[T]}, (S_i[B_k^{\mathbf{X}}])_{k\in[T]})_{i\in\mathcal{C}}$$
$$\mathcal{LW}_r = (L_i[W_r], (L_i[A_t^{W_r}])_{t\in[T]}, (S_i[B_k^{W_r}])_{k\in[T]})_{i\in\mathcal{C}}$$
$$\mathcal{G}_r = ((S_i[f(X_k, W_r)])_{k\in[K]}, (S_i[\xi_k])_{k\in[P]\setminus[K]}, (L_i[\mathbf{A}_t^{\mathbf{G},r}])_{t\in[T]})_{i\in\mathcal{C}}$$

The result follows essentially by observing the dependency structure of the groupings.

As an example, we consider the bundles $\mathcal{LW}_r$ and $\mathcal{SW}_r$ for some $r \in [R]$. The dependencies between the two are represented by the graphical model in Figure 5. From the graphical model we decompose the joint distribution as follows:

$$\begin{aligned}&\mathbf{P}(\mathcal{LW}_r, \mathcal{SW}_r, W_r, (A_t^{W_r})_{t\in[T]})\\ =&\mathbf{P}(W_r)\mathbf{P}((A_t^{W_r})_{t\in[T]}) \times \mathbf{P}(\mathcal{LW}_r|W_r, (A_t^{W_r})_{t\in[T]})\\ &\times \mathbf{P}(\mathcal{SW}_r|W_r, (A_t^{W_r})_{t\in[T]}).\end{aligned}$$

We observe that for each $t = 1, \ldots, T$, $\mathcal{LW}_r$ contains no more than T Lagrange encodings of $A_t^{W_r}$, and hence $\mathcal{LW}_r$ is independent of $(A_t^{W_r})_{t\in[T]}$. Moreover, since $\mathcal{LW}_r$ contains no more than T Lagrange encodings of W_r it is independent of W_r. In particular, these two results imply

$$\mathbf{P}(\mathcal{LW}_r|W_r, (A_t^{W_r})_{t\in[T]}) = \mathbf{P}(\mathcal{LW}_r).$$

Combining with the previous expansion, we have

$$\begin{aligned}&\mathbf{P}(\mathcal{LW}_r, \mathcal{SW}_r, W_r, (A_t^{W_r})_{t\in[T]})\\ =&\mathbf{P}(W_r)\mathbf{P}((A_t^{W_r})_{t\in[T]})\mathbf{P}(\mathcal{LW}_r)\mathbf{P}(\mathcal{SW}_r|W_r, (A_t^{W_r})_{t\in[T]})\\ =&\mathbf{P}(W_r)\mathbf{P}(\mathcal{LW}_r)\mathbf{P}(\mathcal{SW}_r, (A_t^{W_r})_{t\in[T]}|W_r).\end{aligned}$$

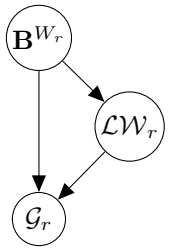

Fig. 6. Dependencies between $\mathcal{LW}_r$ and $\mathcal{G}_r$, where $\mathbf{B}^{W_r}$ is shorthand for $(B_k^{W_r})_{k\in[K+T]\setminus[K]}$

Marginalizing the above expression over $(A_t^{W_r})_{t\in[T]}$ yields

$$\begin{aligned}\mathbf{P}(\mathcal{LW}_r, \mathcal{SW}_r, W_r) &= \mathbf{P}(W_r)\mathbf{P}(\mathcal{LW}_r)\mathbf{P}(\mathcal{SW}_r|W_r)\\ &= \mathbf{P}(W_r)\mathbf{P}(\mathcal{LW}_r)\mathbf{P}(\mathcal{SW}_r),\end{aligned}$$

where the last equality follows because $\mathcal{SW}_r$ is independent of W_r.

Followng the same reasoning, we can factorize $\mathbf{P}(\mathcal{LX}, \mathcal{SX}, X)$ as

$$\mathbf{P}(\mathcal{LX}, \mathcal{SX}, X) = \mathbf{P}(X)\mathbf{P}(\mathcal{LX})\mathbf{P}(\mathcal{SX}).$$

The same strategy applies concerning the bundles $\mathcal{LW}_r$ and $\mathcal{G}_r$. In this case, the relevant graph is displayed in Figure 6. The dependence structure allows the joint distribution to decompose as

$$\begin{aligned}&\mathbf{P}(\mathcal{G}_r, \mathcal{LW}_r, (B_k^{W_r})_{k\in[K+T]\setminus[K]})\\ =&\mathbf{P}((B_k^{W_r})_{k\in[K+T]\setminus[K]}) \times \mathbf{P}(\mathcal{LW}_r|(B_k^{W_r})_{k\in[K+T]\setminus[K]})\\ &\times \mathbf{P}(\mathcal{G}_r|\mathcal{LW}_r, (B_k^{W_r})_{k\in[K+T]\setminus[K]}).\end{aligned}$$

However, since $\mathcal{G}_r$ contains no more than T secret shares of $f(B_k^{\mathbf{X}}, B_k^{W_r})$ for each $k \in [K+T]\setminus[K]$, it is independent of $(B_k^{W_r})_{k\in[K+T]\setminus[K]}$. Moreover, since $\mathcal{LW}_r$ contains no more than T Lagrange encodings of W_r and less than T secret shares of the pads used to Lagrange encode W_r, it is independent of $(W_r, (\xi_k)_{k\in[K+T]\setminus[K]})$. In particular, chaining these two results, we have

$$\mathbf{P}(\mathcal{G}_r|\mathcal{LW}_r, (B_k^{W_r})_{k\in[K+T]\setminus[K]}) = \mathbf{P}(\mathcal{G}_r)$$

Combining with the previous expression, we find that

$$\mathbf{P}(\mathcal{G}_r, \mathcal{LW}_r, (B_k^{W_r})_{k\in[K+T]\setminus[K]}) = \mathbf{P}(\mathcal{LW}_r, (B_k^{W_r})_{k\in[K+T]\setminus[K]})\mathbf{P}(\mathcal{G}_r).$$

Marginalizing the above expression over $(B_k^{W_r})_{k\in[K+T]\setminus[K]}$ yields

$$\mathbf{P}(\mathcal{G}_r, \mathcal{LW}_r) = \mathbf{P}(\mathcal{G}_r)\mathbf{P}(\mathcal{LW}_r)$$

While we do not show the full dependency graph here for the sake of space, we can continue this same manner, ultimately showing that the joint distribution fully factorizes over the groups of messages,

$$\begin{aligned}&\mathbf{P}(\mathbf{X}, W, \mathcal{SX}, \mathcal{LX}, (\mathcal{SW}_r, \mathcal{LW}_r, \mathcal{G}_r)_{r\in[R]})\\ =&\mathbf{P}(\mathbf{X})\mathbf{P}(W)\mathbf{P}(\mathcal{SX}, \mathcal{LX}, (\mathcal{SW}_r, \mathcal{LW}_r, \mathcal{G}_r)_{r\in[R]}),\end{aligned} \quad (46)$$

which demonstrates the desired independence. □

APPENDIX C
COMPLEXITY ANALYSIS

In this section, we show how we arrive at our complexity results shown in Section VI for DeepPrivate and BGW. We analyze the complexity of these two protocols when training a DNN using layer-by-layer training and polynomial activations.

Workers: We've shown this result for our protocol in the decoding step. For BGW, since the workers operate directly on secret shares, they need enough workers to recover a secret share of a multiplication gate. Thus, $N \geq 2T+1$.

Computation: Recall that the local computation for the weights W is as follows:

$$\frac{\partial C}{\partial W} = 2(V(\hat{Y} - Y) \circ Z)X^\top \quad (47)$$

Here, V has shape $h \times \ell$, $\hat{Y}$ and Y have shape $\ell \times m$, Z has shape $h \times m$, and X has shape $d \times m$. Note that since each worker does a fixed amount of matrix multiplications, this computation will be dominated by the multiplication with the largest dimension. We assume here that $m >> \ell$. In this case, that is the term $ZX^\top$. In our protocol, X has dimension $\frac{m}{K}$. Thus, this computation is $O(\frac{m}{K}dh)$. For BGW, this is $O(mdh)$. Since this computation is done every iteration, we multiply each term by R. Similarly, the computation for $\frac{\partial C}{\partial V}$ incurs a computation complexity of $O(\frac{m}{K}\ell h)$ for DeepPrivate and $O(m\ell h)$ for BGW.

Communication: We first take BGW and examine the communication complexity for a given iteration. Adopting the technique proposed in [6] to improve the communication efficiency of the vanilla BGW, after every multiplication, each worker broadcasts its result perform a degree-reduction operation. Since they are computing a fixed amount of multiplication gates, the complexity is determined by the largest item they must send to each other. When doing the forward pass to compute $\frac{\partial C}{\partial W}$, the workers must compute $Z = WX$ with shape $h \times m$. They must also compute $ZX^\top$ with shape $h \times d$. Thus, each node must broadcast matrices of these sizes to other nodes. Similarly, when computing $\frac{\partial C}{\partial W}$, each worker needs to broadcast matrices of sizes $\ell \times m$ and $\ell \times h$. So the complexity for one iteration is $O(hmR + hdR + \ell mR + \ell hR)$ broadcasts per worker.

DeepPrivate, on the other hand, incurs communication costs for encoding and exchanging local computation results. When encoding the data, workers send matrices of size $\frac{md}{K}$ to each other worker. Each iteration when encoding the weights, workers send matrices of size dh and ℓh to each worker. Thus, the encoding communication complexity is $O(\frac{md}{K}N + h(d+\ell)NR)$.

Encoding/Decoding Computation: In DeepPrivate, the workers incur the computational work of encoding and decoding as a result of transitioning in and out of the Lagrange domain. In the BGW scheme, as all computations are done over secret shares, workers don't incur this cost.

In the encoding step, worker i must do three things: interpolate a Lagrange polynomial, evaluate it at N points, and

then interpolate the points it receives from the other workers. Creating a Lagrange polynomial using $K+T$ points has complexity $O((K+T)\log(K+T))$. Evaluating this polynomial at N points takes $O(N\log N)$ time. Interpolating a polynomial of degree T with $T+1$ points takes $O((T+1)\log(T+1))$ time. We then multiply by the size of the interpolation points: $\frac{md}{K}$. Since $N \geq (K+T)$, the complexity of encoding the is data $O(\frac{mdN\log N}{K})$. The complexity of encoding the weights is similarly computed as $O(h(d+\ell)(N\log N)R)$.

In the decoding phase, worker i must evaluate a degree $9(K+T-1)+1$ polynomial at N points, and then interpolate a polynomial using $9(K+T-1)+1$ points. Each point has size $h \times d$. Thus, the complexity of the decoding step is $O(h(d+\ell)(N\log N)R)$.

APPENDIX D
QUANTIZATION

We detail the quantization procedure that is referenced in Section IV-A. The data owners undergo this procedure a single time at the beginning of the protocol.

The quantization procedure consists of a rounding operation applied to the scaled data. We define the element-wise rounding operation $Round(X)$ by

$$[Round(X)]_{ij} = \begin{cases} \lfloor X_{ij} \rfloor & \text{if} \quad X_{ij} - \lfloor X_{ij} \rfloor < 0.5 \\ \lfloor X_{ij} \rfloor + 1 & \text{otherwise} \end{cases} \tag{48}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x. We also employ $\phi : \mathbb{Z} \to \mathbb{F}_p$, defined by

$$\phi(x) = \begin{cases} x & \text{if } x \geq 0 \\ p+x & \text{if } x < 0 \end{cases} \tag{49}$$

to suitably represent a negative integer in the finite field by using two's complement representation.

Data node $i \in [M]$ quantizes its data by applying $X_i \leftarrow \phi\left(Round(2^{l_x} \cdot X_i)\right)$, where l_x is an integer parameter to control the quantization loss.