Private, yet Practical, Multiparty Deep Learning

Xinyang Zhang  
Lehigh University  
xicz15@lehigh.edu

Shouling Ji  
Zhejiang University  
sji@zju.edu.cn

Hui Wang  
Stevens Institute of Technology  
hui.wang@stevens.edu

Ting Wang  
Lehigh University  
inbox.ting@gmail.com

Abstract—In this paper, we consider the problem of multiparty deep learning (MDL), wherein autonomous data owners jointly train accurate deep neural network models without sharing their private data. We design, implement, and evaluate ∞MDL, a new MDL paradigm built upon three primitives: asynchronous optimization, lightweight homomorphic encryption, and threshold secret sharing. Compared with prior work, ∞MDL departs in significant ways: a) besides providing explicit privacy guarantee, it retains desirable model utility, which is paramount for accuracy-critical domains; b) it provides an intuitive handle for the operator to gracefully balance model utility and training efficiency; c) moreover, it supports delicate control over communication and computational costs by offering two variants, ∞MDL\text{L} \text{L} and ∞MDL\text{L}, operating under loose and tight coordination respectively, thus customizable for given system settings (e.g., limited versus sufficient network bandwidth). Through extensive empirical evaluation using benchmark datasets and deep learning architectures, we demonstrate the efficacy of ∞MDL.

I. INTRODUCTION

The recent advances in deep learning (DL) \cite{DL} have led to breakthroughs in long-standing artificial intelligence tasks (e.g., image recognition, language translation, and even playing Go \cite{Go}), enabling use cases previously considered strictly experimental. Such success is premised on the availability of massive training data. Most deployed DL systems are built upon large, centralized data repositories. This paradigm abounds with privacy risks: often, data owners neither understand nor have control over how their private data is being used. Moreover, in a range of domains, notably those related to healthcare, the sharing of personal information is forbidden by regulations. Thus, following the centralized-training paradigm, clinical sites and institutions can only perform analysis over the high-level design of MDL, which introduces the building blocks of MDL; Section \text{III} presents the high-level design of MDL; Section \text{IV} and \text{V} elaborate the two variants of MDL, followed by their empirical evaluation in Section \text{VI}; Section \text{VII} surveys relevant literature; the paper is concluded in Section \text{VIII}.

II. PRELIMINARIES

A. Deep Learning

Deep learning (DL) represents a class of machine learning algorithms which learn high-level abstraction of complex data using multiple processing layers and non-linear transformations. We primarily focus on supervised learning, wherein the training inputs are associated with “labels” while the goal is to learn models to predict the labels for future inputs. Our discussion is applicable to unsupervised learning as well.
To determine its global descent. The update rule for the gradient computed from the current mini-batch. This process is via a back-propagation procedure. The update rule for labels and the model’s outputs) is computed for every mini-batch. Due to the model complexity, stochastic training set. The gradient of each parameter \( \lambda \) on a victim’s privacy at the cost of sabotaging the tasks. For instance, all the malicious parties may empty their training data to expose the victim’s private information in the training outcome. Practically, in healthcare domains, clinical sites are strongly incentivized to collaboratively train predictive models with better demographical coverage and diagnosis accuracy than models trained on their own data; yet, they may also be interested to learn other parities’ data ownerships.

**C. Cryptographic Building Blocks**

Below we introduce two cryptographic primitives that serve as the building blocks of \( \infty \text{MDL} \).

\[ \text{A. Attack Model} \]

Like most other privacy-preserving machine learning frameworks (e.g., [10], [11], [12], [5]), we assume a semi-honest attack model: each party strictly follows the predefined protocol but is curious about the private information of other parties; further, different parties may collude with each other, attempting to infer a victim’s private information.

We remark that this threat model is realistic. Theoretically, it is impossible to prevent malicious parties from invading on a victim’s privacy at the cost of sabotaging the tasks. For instance, all the malicious parties may empty their training data to expose the victim’s private information in the training outcome. Practically, in healthcare domains, clinical sites are strongly incentivized to collaboratively train predictive models with better demographical coverage and diagnosis accuracy than models trained on their own data; yet, they may also be interested to learn other parities’ data ownerships.

\[ \text{B. Attack Model} \]

Training a DNN model (see Figure 1) is to optimally configure its parameters. Due to the model complexity, stochastic gradient descent and its variants are often used [8]. During each “epoch”, a set of “mini-batches” are sampled from the training set. The gradient of each parameter \( w \) with respect to the objective function (e.g., the cross entropy of the training labels and the model’s outputs) is computed for every mini-batch via a back-propagation procedure. The update rule for \( w \) is \( w := w - \lambda \nabla_w \), where \( \lambda \) is the learning rate and \( \nabla_w \) is the gradient computed from the current mini-batch. This process repeats until the objective function converges.

This procedure can be generalized to the case of multiple parties: each party performs training using its own data; at the end of each epoch, they share with each other their “local” gradients; the local gradients of a parameter \( w \) are aggregated to determine its global descent. The update rule for \( w \) is:

\[ \text{Training a DNN model (see Figure 1) is to optimally configure its parameters. Due to the model complexity, stochastic gradient descent and its variants are often used [8]. During each “epoch”, a set of “mini-batches” are sampled from the training set. The gradient of each parameter } w \text{ with respect to the objective function (e.g., the cross entropy of the training labels and the model’s outputs) is computed for every mini-batch via a back-propagation procedure. The update rule for } w \text{ is } w := w - \lambda \nabla_w, \text{ where } \lambda \text{ is the learning rate and } \nabla_w \text{ is the gradient computed from the current mini-batch. This process repeats until the objective function converges.} \]

This procedure can be generalized to the case of multiple parties: each party performs training using its own data; at the end of each epoch, they share with each other their “local” gradients; the local gradients of a parameter \( w \) are aggregated to determine its global descent. The update rule for \( w \) is:

\[ w := w - \lambda \sum_{A_i \in A} \nabla_w \]

(1)

where \( A \) denotes the set of parties, \( A_i \) is the \( i \)-th party, and \( \nabla_w \) is \( w \)'s gradient computed using the data owned by \( A_i \).

Despite its efficiency [8], [9], this construction is potentially privacy-violating: the local gradients are exploitable to reveal sensitive information of contributing parties [3]. [4]. We thus consider local gradients as private information to be protected.

\[ \text{B. Attack Model} \]

Like most other privacy-preserving machine learning frameworks (e.g., [10], [11], [12], [5]), we assume a semi-honest attack model: each party strictly follows the predefined protocol but is curious about the private information of other parties; further, different parties may collude with each other, attempting to infer a victim’s private information.

We remark that this threat model is realistic. Theoretically, it is impossible to prevent malicious parties from invading on a victim’s privacy at the cost of sabotaging the tasks. For instance, all the malicious parties may empty their training data to expose the victim’s private information in the training outcome. Practically, in healthcare domains, clinical sites are strongly incentivized to collaboratively train predictive models with better demographical coverage and diagnosis accuracy than models trained on their own data; yet, they may also be interested to learn other parities’ data ownerships.

\[ \text{C. Cryptographic Building Blocks} \]

Below we introduce two cryptographic primitives that serve as the building blocks of \( \infty \text{MDL} \).
via the global model at $M$. It is noted that once the training is complete (i.e., the global model converges), each worker can independently and privately evaluate the trained model over new data, without interacting with other workers.

B. Basic Design

Now we incorporate proper PEMs into the basic construction. One straightforward solution is to perform differentially private randomization [6] to the local gradients, with the hope that releasing such local gradients does not overly leak the private information of the local data [18], [5], which we refer to as the local differential privacy (LDP) solution.

Intuitively, a function $f$ is differentially private if its probability of producing a particular output is not sensitive to whether a specific data entry is included in its input: given any two datasets $D, D'$ differing by a single entry and any output $O$, it holds that $\Pr(f(D) \in O) \leq \exp(\epsilon) \cdot \Pr(f(D') \in O)$, where $\epsilon$ is the “budget” controlling the tolerable privacy loss. We may specify such budget for each parameter and randomize its local gradient using the Laplacian or Gaussian mechanisms [6].

Despite its simplicity, this mechanism suffers low model utility. Specifically, applying DP requires estimating the global sensitivity: how the randomization of the input impacts the global output. However, in MDL, due to lacking the knowledge about others’ data, it is extremely obscure for a worker to estimate the influence of its input to the global model. Therefore, overly conservative noise needs to be injected, resulting in significant utility loss in the global model. Such utility loss is consequential for accuracy-demanding domains [19].

C. Enhanced Design

To mitigate this issue, we integrate the LDP mechanism with a coordination mechanism. Our design is motivated by the following observations. Most DP mechanisms employ random noise sampled from symmetric distributions (e.g., Laplace and Gaussian); as more workers aggregate their local gradients, a larger part of the injected noise can cancel out. Formally,

**Theorem 1.** Let $\nabla i$ denote the local gradient of the $i$-th worker and $r^i$ be the differentially private noise: $r^i \sim \text{Lap}(\frac{1}{n})$. The estimated global gradient $\frac{1}{|A|} \sum_{A_i \in A} (\nabla i + r^i)$ is a random variable with mean $\frac{1}{|A|} \sum_{A_i \in A} \nabla i$ and variance $\frac{\epsilon}{2|A|}$. \[ \square \]

**Proof.** It follows from the definition of global gradient in Eqn. (1) and the independence of random noise $\langle r^i \rangle_{A_i \in A}$.

Intuitively, as more workers aggregate local gradients, we obtain better estimates of global gradients. We formalize this intuition with the concept of $\rho$-visibility: an $n$-worker MDL system is $\rho$-visible, if the global gradient of each parameter is visible (available for download) only after it aggregates the local gradients of at least $n = \rho \cdot n$ workers. The subset of workers contributing to parameter $w$ is called the active set of $w$, denoted by $A_w$. Clearly, $\rho$-visibility enforces $|A_w| \geq \rho \cdot |A|$ for any $w$ at any time.

We may consider $\rho$-visibility as another layer of privacy protection, because it essentially enforces a generalized abstraction of secure aggregation [20]: the global information is revealed only when it has included the local information from a sufficient number of individuals. It is also noted that $\rho$-visibility does not compromise the protection offered by the LDP mechanism, as it is applied in the post-processing stage of LDP. Therefore, they can be integrated in synergy, which we name as the $\times$MDL mechanism.

Next we discuss a few instantiations of $\rho$-visibility to show the tradeoff between model utility and training efficiency.

D. Implications of Rho-Visibility

As more workers aggregate local gradients, we obtain more accurate global gradients, leading to better utility of the global model. However, this improvement is not free: the synchronization among more workers at every epoch implies higher computational and communication costs per epoch. Given the assumption that the global model converges roughly within a fixed number of epochs (which is empirically validated in Section VI), more synchronization results in higher overall training cost. Thus, there exists inherent tradeoff between model utility and training efficiency, as shown in Figure 3.

Concretely, $\rho = 1$ corresponds to a synchronous protocol. After each training epoch, the workers contribute their local gradients to estimating the global gradient. The global gradient is computed in a privacy-preserving manner, which ensures that each worker only learns the global gradient without knowing the local gradients of other workers.

Meanwhile, $\rho = \frac{1}{n}$ entails an asynchronous protocol [5], wherein the workers perform the training in a collaborative yet uncoordinated manner. In specific, after finishing an epoch, each worker asynchronously uploads its local gradient to the manager, downloads the gradients contributed by other workers, and updates its local model.

Finally, $\frac{1}{n} < \rho < 1$ corresponds to a hybrid protocol, which features better model utility than asynchronous protocols and lower training cost than synchronous protocols. Therefore, the operator is able to balance model utility and training efficiency by adjusting $\rho$.

E. $\times$MDL: A Nutshell View

At the core of $\times$MDL is a novel use of lightweight homomorphic encryption and threshold secret sharing to construct a gradient exchange protocol. In a nutshell, each worker $A_i$ is assigned a private key $s_k$, and all the workers share a public key $pk$. For each parameter $w$, $A_i$ encrypts its local gradient $\nabla w$ as $\text{Enc}_{pk} (\exp (\nabla w))$. The multiplicative homomorphism entails: $\prod_i \text{Enc}_{pk} (\exp (\nabla w)) = \text{Enc}_{pk} (\exp (\sum_i \nabla w))$. The aggregated ciphertext thus encodes the summation of local gradients of active workers, which is equivalent to the global
gradient. If at least \( m (m = \rho \cdot n) \) workers have contributed local gradients, the global gradient is automatically decryptable, thereby available for all the workers to access; otherwise, the global gradient is invisible.

Although existing cryptosystems (e.g. [21], [22], [23]) also meet the requirement that more than a threshold number of workers are required to decrypt the ciphertext, our setting differs in significant ways. First, existing schemes implicitly assume that the set of active workers are known in advance. In MDL setting, the active workers are determined on the fly; each worker may contribute to different sets of parameters during different epochs. Second, existing schemes require multiple rounds of communication among the active workers, which is costly for the setting of a large number of parameters and active workers. Finally, exiting schemes are designed to encrypt and decrypt a fixed secret message, while in our target setting, the secret message (i.e., the summation of local gradients) is dynamically constructed by the active workers.

Next we detail two variants of \( \infty \text{MDL} \) operating under tight and loose coordination respectively. The symbols used in the following are summarized in Table I.

### IV. \( \infty \text{MDL} \)

#### A. Overview

In \( \infty \text{MDL} \), the workers are shepherded by the manager to perform training in a semi-synchronous manner. Specifically, \( \infty \text{MDL} \) involves the following operations (without loss of generality, we consider a specific parameter \( w \))

- **Setup** - A trusted authority \( T \) (e.g., certificate authority) generates cryptographic keys, announces the public keys, and distributes the private keys to the workers.
- **Training** - Each worker \( A_i \) (\( i = 1, 2, \ldots, n \)) independently trains the model over its private data and derives the local gradient \( \nabla_w^i \) for each parameter \( w \).
- **Register** - \( A_i \) selects a set of parameters \( R_i \) to contribute, and registers its requests with the manager \( M \).
- **Callback** - If at least \( m \) workers register for \( w \), \( M \) invokes them to upload their local gradients; meanwhile, once the global model is updated, \( M \) notifies these workers to download the latest value of \( w \) (i.e., \( w^\theta \)).
- **Upload** - Invoked by \( M \), \( A_i \) uploads its local gradient \( \nabla_w^i \) to \( M \) in a privacy-preserving manner.

#### B. Protocols

Below we elaborate the operations from the perspective of each key player of \( \infty \text{MDL} \).

1. **Trusted Authority**

   Input: generator \( g \), primes \( p, q \)
   
   ```
   // generate keys
   1. \( s \leftarrow \mathbb{Z}_p \)  
   2. public key \( h \leftarrow g^s \)  
   3. \( \{(x_i, s_i)\}_{i=1}^n \leftarrow \text{Shamir's Protocol with } s \text{ as secret} \)  
   // distribute keys
   4. announce \( h \) and \( \{(x_i, g^{s_i})\}_{i=1}^n \)  
   5. for \( i = 1, 2, \ldots, n \) do assign \( s_i \) to \( A_i \)  
   ```

2. **Manager**

   ```
   // manager
   // manager interactions, as shown in Figure 4. Also note that we assume a simple contribution-based download policy, that is, each worker is only allowed to download the global parameters which it contributes to. We consider developing more advanced download policies as our ongoing research.
   ```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>total number of workers in MDL ( \infty )</td>
</tr>
<tr>
<td>( \rho )</td>
<td>visibility threshold</td>
</tr>
<tr>
<td>( m )</td>
<td>minimum size of active set ( \rho \cdot n )</td>
</tr>
<tr>
<td>( t )</td>
<td>total number of parameters in DNN</td>
</tr>
<tr>
<td>( k )</td>
<td>number of parameters registered by each worker</td>
</tr>
<tr>
<td>( k^* )</td>
<td>minimum ( k ) required for MDL ( \infty ) to function</td>
</tr>
<tr>
<td>( A_w )</td>
<td>active set of workers with respect to ( w )</td>
</tr>
<tr>
<td>( \nabla_w )</td>
<td>local gradient of ( w ) by worker ( A_i )</td>
</tr>
<tr>
<td>( R_i )</td>
<td>set of parameters registered by ( A_i )</td>
</tr>
<tr>
<td>( c_i )</td>
<td>ciphertext of ( \nabla_w )</td>
</tr>
<tr>
<td>( z_i )</td>
<td>Lagrange coefficient of ( A_i ); ( z_i = \prod_{j=1 \atop j \neq i}^{n} \frac{x_j - x_i}{x_j - x_i} )</td>
</tr>
<tr>
<td>( \nabla_w^g )</td>
<td>global gradient of ( w )</td>
</tr>
<tr>
<td>( r_i )</td>
<td>random nonce of ( A_i )</td>
</tr>
<tr>
<td>( r )</td>
<td>sum of random nonce ( \sum_{A_i \in A_w} r_i )</td>
</tr>
</tbody>
</table>

Table I. Symbols and notations.

**Download -** Notified by \( M \), \( A_i \) downloads the global version of \( w (w^\theta) \) and applies it to its local model.

![Figure 3: Tradeoff of model utility and training efficiency.](image)

![Figure 4: Protocols of worker-manager interaction.](image)
Algorithm 2: Manager $M$

Input: public key $h$, generator $g$, public shares $\{g^{\cdot i}, x_i\}_{i=1}^n$

1. /* initialization */
2. initialize global parameters;
3. while approximate optimum is not achieved yet do
   4. /* request (register) */
   5. event register by $A_t$ to contribute to $w$
      6. add $A_t$ to $A_w$;
   7. /* callback (upload) */
   8. event callback $A_w$ to upload $w$
      9. receive $c_i, g^{c_i}$ from $A_i$;
      10. for $A_i \in A_w$ do
          11. $g^{c_i} \leftarrow g^{\sum \lambda_i g_i} \prod A_i$;
      12. receive $g^s$ to $A_i$;
      13. receive $d_i \leftarrow (g^{s_i} r_i)$;
      14. // decrypt global gradient
      15. compute $\nabla^g_w = \frac{1}{|A_w|} \ln g^{s_i} x_i ;
      16. // update global parameter
      17. update $w^g \leftarrow w^g = \lambda \nabla^g_w$;
      18. /* callback (download) */
      19. event callback $A_w$ to download $w$
      20. for $A_i \in A_w$ do send $w^g$ to $A_i$;

where $r = \sum A_i \cdot r_i$.

Then for $|A_w| \geq m$, based on the property of Shamir's protocol, we have

$$\prod_{A_i \in A_w} d_i = \prod_{A_i \in A_w} g^{s_i z_i} = g^{\sum A_i \cdot s_i x_i} = g^r$$

Thus, $\nabla^g_w = \frac{1}{|A_w|} \ln (\prod A_i \cdot c_i) = \frac{\sum A_i \cdot \nabla^g_w}{|A_w|}$.

Further, we show the overall correctness of the protocols above. Particularly, we only need to prove:

**Theorem 3.** During any training epoch, the parameter $w$ is correctly updated by the worker-manager interaction.

*Proof. (Sketch)* During each epoch, from the perspective of either participating worker $A_t$ or manager $M$, the protocol of updating $w$ is strictly serial, as shown in Figure 4 consisting of register, callback (upload), upload, callback (download) and download. This serialism entails that $A_t$ and $M$ always send and receive the correct version of $w$.

In current implementation, the worker $A_t$ randomly selects $k$ parameters as $R_i$. It can derived that the probability of a given parameter missing updates (i.e., less than $m$ workers register for it) in an epoch is at most $(n-m+1)(t-k)^{-m+1}$, where $t$ is the total number of parameters. In realistic settings, a small $k$ often suffices: for example, with $n = 8$ and $m = 2$, $k = 0.47 \cdot t$ ensures that the missing rate is below 10%. Also note that the minimum $k$ ($k^*$) necessary for each parameter to be updated is $\lceil m \cdot t/n \rceil$.

**b) Manager:** After initializing the global model (line 1), $M$ handles all requests and callbacks (line 2-13). It manages an active list $A_w$ for $w$, which records the workers registered to contribute to $w$. On receiving the register request by $A_t$ for $w$, $M$ adds $A_t$ to $A_w$ (line 4). Once all the workers finish registration, $M$ calls back the workers in $A_w$ and receives from them encrypted local gradients, one-time nonces and auxiliary information to update $w$ (line 6-9). In the event that $w$ is ready for download, $M$ notifies the workers in $A_w$ to receive the latest value of $w^g$ (line 13).

**c) Worker:** After initializing its local model, $A_t$ enters the training loop. During each training epoch, $A_t$ first decides the parameters to which it intends to contribute ($R_i$) and registers $R_i$ with $M$ (line 3). We will discuss the selection of $R_i$ shortly. Then, $A_t$ runs SGD over its private data and updates local parameters (line 4-5). Next, $A_t$ enters an event-driven loop (line 6-14). In the event of callback by $M$ to update $w$, $A_t$ uploads the encrypted local gradient $c_i$, an encrypted one-time nonce $r_i$ and auxiliary information to $M$ (line 8-11); in the event of callback to download $w$, $A_t$ updates $w$ with its global version (line 13-14).

**C. Analysis**

Next we analyze the correctness, privacy and complexity properties of $\alpha$MDL$^c$.

**a) Correctness:** We show that during each epoch, the global gradient of parameter $w$ can be correctly decoded, if at least $m$ workers contribute to $w$.

**Theorem 2.** If $|A_w| \geq m$, then the global gradient of $w$ is correctly decryptable as $\frac{1}{|A_w|} \sum A_i \cdot \nabla^g w$.

**Proof.** From the definitions of encryption procedure and multiplicative homomorphism, we have

$$\prod A_i \cdot c_i = g^{sr} \exp \left( \sum A_i \cdot \nabla^g_w \right)$$

Algorithm 3: Worker $A_i$

Input: secret share $s_i$, primes $p$, $q$

1. /* initialization */
2. initialize local parameters;
3. while approximate optimum is not achieved yet do
   4. /* $A_i$ registers to contribute to $w$ */
   5. register requests $R_i$ with $M$;
   6. /* training */
   7. run Sitz on local dataset to compute $\{\nabla^g_w\}_w$;
   8. for $w \notin R_i$ do update $w^g \leftarrow w^g - \lambda \nabla^g_w$;
   9. while $R_i \neq \emptyset$ do
      10. /* upload */
      11. event callback by $M$ to upload $w$
      12. // local gradient, one-time nonce
      13. $c_i \leftarrow \exp(\nabla^g_w) \cdot h^{r_i} / r_i \in Z_q$;
      14. send $c_i, g^{r_i}$ to $M$;
      15. // auxiliary information
      16. receive $g^{r_i}$ from $M$;
      17. compute and send $(g^{r_i})^{r_i}$ to $M$;
      18. /* download */
      19. event callback by $M$ to download $w$
      20. receive $w^g$ and update $w^g \leftarrow w^g$;
      21. remove $w$ from $R_i$.
Theorem 4. If \(|A_w| < m\), the global gradient \(\nabla_w^g\) of \(w\) is undecryptable.

Proof. (Sketch) We use the concepts of zero-knowledge and simulatability [24]. The system involves the following players: \(A_w\), the manager \(M\) and the rest workers (collectively denoted by \(B_w\)). Given \(\{\nabla_w^g, c_t, d_t, g^{r_t}\}_{A_w}\), it is straightforward to see that the interaction between \(A_w\) and \(B_w\) can be simulated; more precisely, the probability distribution of the views is simulatable. Thus, no extra information is revealed about \(\{s_i, r_i\}_{A_w}\) other than \(\{c_i, d_i, g^{r_i}\}_{A_w}\).

To reveal \(\nabla_w^g\) from \(\{c_i\}_{A_w}\), the attacker needs to solve \(g^{r_i}\). Given the accessible information, this is equivalent to (i) computing \(g^{st}\) from \(g^s\) and \(g^t\) (guarded by the hardness of computational Diffie-Hellman problem), (ii) computing \(g^{r_t}\) from \(\{d_t\}_{A_w}\) for \(|A_w| < m\) (guarded by the Shamir’s protocol), or (iii) computing \(s\) (or \(r\)) from \(g^s\) (or \(g^r\)) (guarded by the hardness of discrete logarithm). In other words, an attacker that reveals \(\nabla_w^g\) implies an attacker that can efficiently solve one of the problems above.

Further, we show the privacy guarantee for local gradients. We consider the worst case that all other workers collude with each other, attempting to reveal the local gradient of a victim.

Theorem 5. If \(|A_w| < m\), the local gradient of an individual worker in \(A_w\) is undecryptable, even if all other workers in \(A_w\) collude with each other.

Proof. (Sketch) We again use the concept of zero-knowledge proof. Without loss of generality, let \(A_i\) be the victim and \(A_i'\) be the remaining parties in the system. Given \(\{\nabla_i^g, c_t, d_t\}_{A_i}\), it is trivial to see that the interaction between \(A_i\) and \(A_i'\) can be simulated. No extra information is leaked about \(s_i, r_i\) other than \(c_i, d_i\). Thus, if the computational Diffie-Hellman problem is hard, the computation of \(\nabla_i^g\) is also hard.

c) Complexity: Since the local training often dominates each worker’s computation load (see our empirical evaluation), we focus our analysis on the communication complexity. In particular, we consider the case that each worker registers for the minimum number of parameters \((k^* = \lceil m \cdot t/n \rceil\)). In Algorithm 3 during each epoch, for each \(w \in R_t\), \(A_i\) exchanges a constant number of integers with \(M\), which entails the overall communication cost of \(O(\lceil m \cdot t/n \rceil)\). Correspondingly, in Algorithm 2 the communication cost of \(M\) is the aggregated costs of all workers, i.e., \(O(t \cdot m)\).

V. \(\infty\text{MDL}^d\)

A. Overview

At a high level, \(\infty\text{MDL}^d\) departs from \(\infty\text{MDL}^c\) in that it requires little coordination among the participating workers, thereby enabling high concurrency. In specific, for given parameter \(w\), \(\infty\text{MDL}^c\) requires fixing the set of active workers \(A_w\) (i.e., the regiseter operation) before allowing them to upload the local gradients of \(w\). In contrast, in \(\infty\text{MDL}^d\), \(A_w\) is constructed on the fly; the workers contribute to \(w\) on a voluntary and first-come-first-served basis.

In \(\infty\text{MDL}^c\), \(M\) updates \(w\) only if all the workers in \(A_w\) have committed their local gradients. This restriction results in costly delay in each epoch in cases of (i) large-size \(A_w\) or (ii) the workers with imbalanced computation capacities. Also it may cause the robustness issue, as the malfunction of a single worker in \(A_w\) causes the failure of updating \(w\). In \(\infty\text{MDL}^d\), once any \(m (m = \rho \cdot n)\) workers have uploaded their local gradients of \(w\), \(M\) is able to update \(w\) immediately. This difference entails \(\infty\text{MDL}^d\)’s significant advantage over \(\infty\text{MDL}^c\) in execution efficiency and fault tolerance.

Algorithm 4: Manager \(M\)

Input: public key \(h\), generator \(g\), public shares \(\{g^s, x_i\}_{i=1}^n\)
// initialization \(\{\ldots\}\) 
1 while approximate optimum is not achieved yet do
   /* request (upload) */
   2 /* c-value */
      receive \(c_t\) and \(g^{r_t}\) from \(A_i\);
   3 /* z- and d-values */
      for \(A_j \in A_w\) do
         update \(z_j \leftarrow z_j \cdot g^{r_j}\);
         send \(g^{x_j} z_j\) to \(A_i\) and receive \(g^{x_j} z_j\);
      update \(d_t \leftarrow (d_t) g^{x_j} z_j\);
      add \(A_i\) to \(A_w\);
      assign \(z_i \leftarrow \prod_{A_j \in A_w, j \neq i} \frac{x_i}{x_j} z_j\);
      send \(g^{x_j} z_j\) to \(A_i\) and receive \(g^{x_j} z_j\) and assign \(d_t \leftarrow g^{x_j} z_j\);
   6 if \(|A_w| \geq m\) then
      /* decrypt global gradient */
      compute \(\nabla_w^g = \sum_{A_j \in A_w} \log\left(\frac{\prod_{A_j \in A_w} \frac{g^{x_j} z_j}{x_j}}{\prod_{A_j \in A_w} \frac{g^{x_j} z_j}{x_j}}\right)\);
      /* update global parameter */
      update \(w^g \leftarrow w^g - \lambda \nabla_w^g\);
   9 /* callback (download) \{\ldots\} */

B. Protocols

Next we detail the protocols of \(\infty\text{MDL}^d\). For space limitations, we omit the operations identical to \(\infty\text{MDL}^c\).

a) Manager: We focus on the operation of request (upload), with which \(M\) handles the requests by active workers to upload their local gradients.

In specific, \(A_w\) (for parameter \(w\)) maintains the current set of active workers which have uploaded their encrypted local gradients (c-values) of \(w\). Meanwhile, \(M\) maintains two values (z- and d-values) for each worker in \(A_w\). As a new worker \(A_i\) comes, besides receiving its local gradient and one-time nonce (line 3), \(M\) interacts with \(A_i\) and updates z- and d-values for each worker (including \(A_i\)) in \(A_w\) (line 4-10). Once \(m\) workers have uploaded their local gradients, the global gradient \(\nabla_w^g\) is immediately decryptable by aggregating c- and d-values of workers in \(A_w\) (line 12). After update according to Eqn. (11), \(w^g\) is available for workers in \(A_w\) to download (line 13).

b) Worker: We focus on the operation of upload, with which the worker \(A_i\) uploads its local gradient to \(M\). Specifically, \(A_i\) first encrypts its local gradient and sends the ciphertext (c-value) and one-time nonce to \(M\) (line 3). Then \(A_i\) interacts with \(M\) and helps update the z- and d-values corresponding to the workers (including itself) in the current active list \(A_w\) (line 4-7).
Algorithm 5: Worker $A_i$

| Input: secret share $s_i$, primes $p, q$  |
| /\ * initialization \{\ldots\} */ |
| 1 while approximate optimum is not achieved yet do  |
|   /\ training \{\ldots\} */  |
|   /* upload */  |
| 2 for $w \in \mathcal{R}_i$ do  |
|   // e-value, one-time nonce  |
|   $c_i \leftarrow \exp(\nabla_w^i) \cdot h^r_i; r_i \sim \mathbb{Z}_q$; send $c_i, g^{r_i}$ to $M$;  |
|   // d- and z-values  |
|   receive $(g^{\delta_i})_{A_i \in A_w}$ from $M$;  |
|   compute and send $(g^{\gamma_i}_j x_j^{r_i})_{A_j \in A_w}$ to $M$;  |
|   receive $g^{r_i}$ from $M$;  |
|   compute and send $g^{r_i} z_i^{r_i}$ to $M$;  |
|   remove from $\mathcal{R}_i$, parameters $A_i$ has no contributes to;  |
| 3 /* download \{\ldots\} */  |

C. Analysis

We now analyze the correctness, privacy and complexity properties of $\propto M D L^d$. We also compare the strengths and weaknesses of $\propto M D L^f$ and $\propto M D L^d$.

a) Correctness: We first introduce the following lemmas.

**Lemma 1.** If $|A_w| \geq m$, $\prod_{A_i \in A_w} (g^{s_i}) z_i = g^s$.

**Proof.** It can be verified that for $A_i \in A_w$, its $z$-value $z_i = \prod_{A_j \in A_w} (g^{s_j})^{-x_j}$. Given the property of Shamir's protocol, secret $s$ can be recovered by any subset (of cardinality at least $m$) of its shares, i.e., $s = \sum_{A_i \in A_w} s_i z_i$, which leads to $\prod_{A_i \in A_w} (g^{s_i}) z_i = g^s$.

**Lemma 2.** For $A_i \in A_w$, $d_i = g^{s_i} z_i$.

**Proof.** We prove this lemma using induction. Suppose that it holds for any $A_w$ with $|A_w| \leq l - 1$ and that $A_i$ is the $l$-th worker joining $A_w$.

After the interaction between $A_i$ and $M$, for $A_j \in A_w$ ($j \neq i$), we have (we use $x$ and $x'$ to differentiate object $x$ before and after $A_i$ is included in $A_w$):

$$d'_j = (d_j)^{-x_j z_i} \cdot (g^{s_j}) z_i, g^{s_i} z_i = g^{s_j} z_j^{r_j}, g^{s_j} z_j^{r_j}, g^{s_j} z_j^{r_j}$$

Further, by construction, we have $d'_i = g^{s_i} z_i$.

Based upon these lemmas, we have the following theorem.

**Theorem 6.** If $|A_w| \geq m$, the global gradient of $w$ is decryptable: $\nabla_w^g = \frac{1}{|A_w|} \ln \left( \prod_{A_i \in A_w} c_i \right)$.

**Proof.** From the definition of encryption procedure, we have

$$\prod_{A_i \in A_w} c_i = \exp \left( \sum_{A_i \in A_w} \nabla_w^i \right) g^{s_i}$$

Then for $|A_w| \geq m$, given Lemma 1 and 2 and the property of Shamir's protocol, we have

$$\prod_{A_i \in A_w} d_i = \prod_{A_i \in A_w} g^{s_i} z_i = g^{\sum_{A_i \in A_w} s_i z_i} = g^{s}$$

leading to $\nabla_w^g = \frac{1}{|A_w|} \ln \left( \prod_{A_i \in A_w} c_i \right) = \frac{1}{|A_w|} \ln \left( \prod_{A_i \in A_w} d_i \right)$.

The proof of the overall correctness of the protocols is similar to Theorem 3 which we omit here.

b) Privacy: Similar to the analysis in Section IV, we demand privacy assurance for both global and local gradients. First, we have the following theorem that guards the privacy of global gradients.

**Theorem 7.** If $|A_w| < m$ (i.e., less than $m$ workers contribute local gradients), the global gradient $\nabla_w^g$ is undecryptable.

To show the privacy protection for local gradients, again, we assume all other parties collude with each other, attempting to reveal the local gradient of a victim worker.

**Theorem 8.** In $\propto M D L^d$, if $|A_w| < m$, the local gradient of any individual worker $A_i \in A_w$ is undecryptable, even if all other workers in $A_w$ collude with each other.

**Proof.** The proofs of Theorem 7 and 8 are respectively similar to Theorem 4 and 5.

c) Complexity: We consider the case that each worker contributes to a minimum number of parameters (i.e., $k^* = \lceil m \cdot t/n \rceil$); each parameter has exactly $m$ contributors. In Algorithm 5 during each epoch, to update given parameter $w$, the worker $A_i$, assumed to arrive as the $l$-th worker in $A_w$, needs to exchange and update $O(l)$ (i.e., on average, $O(m/2)$) integers with $M$. Thus, the average communication complexity for each worker is $O\left(\lceil m^2 \cdot t/n \rceil\right)$. In Algorithm 4 the communication cost of $M$ is the aggregated communication cost across all participating workers, i.e., $O(m^2 \cdot t)$.

d) Comparison of $\propto M D L^f$ and $\propto M D L^d$: Compared with $\propto M D L^f$, $\propto M D L^d$ requires less coordination, thereby achieving higher execution efficiency. Nevertheless, this advantage is not free. $\propto M D L^d$ features the communication complexity of $O(m^2 \cdot t)$, in contrast of $O(m \cdot t)$ in $\propto M D L^f$, which can be expensive for the settings of limited network bandwidth or a large number of participating workers.

VI. EMPIRICAL EVALUATION

Next using benchmark datasets and DNN architectures, we empirically assess the performance of individual components and overall system of $\propto M D L$; we also explore the strengths and limitations of variants of $\propto M D L$.

A. Experiment Setting

Our experiments used two benchmark datasets, the MNIST1 and SVHN2 datasets. The former comprises 60K training and 10K testing ($28 \times 28$ grayscale) images, while the latter constitutes 73K training and 26K testing ($32 \times 32$ RGB) images. We consider MNIST and SVHN respectively representing “simple” and “hard” DL tasks. Using two disparate datasets, we intend to capture the impact of data dimensionality over the system performance. The private data of each worker comprises 60% random samples from the training dataset.

1 http://yann.lecun.com/exdb/mnist
2 http://ufldl.stanford.edu/housenumbers
We implemented all alternative MDL solutions on Theano, one of the most popular DL platforms. We considered two major DNN architectures adapted from [5], multi-layer perception (MLP) and convolutional neural network (CNN), both of which have been widely used in image recognition tasks. We regard the CNN and MLP models respectively representing “strong” and “weak” DL models.

We contrasted the performance of alternative MDL frameworks using three metrics, model utility, convergence rate and training efficiency. Specifically, the model utility is measured by its classification accuracy over the testing data; the convergence rate is captured by the number of epochs necessary for the model to converge; and the training efficiency is measured by the end-to-end training time of the model.

The default parameter setting is as follows: the number of workers $n = 8$, the visibility threshold $\rho = 0.5$, the learning rate $\lambda = 0.01$, and the mini-batch size 128. Each worker is running a 3.40 GHz processor and 16 GB RAM.

B. Experiment Results

a) Encryption and Decryption: We start by assessing the overhead of the cryptographic machinery used by $\infty$MDL.

Figure 5(a) shows how the execution time of the encryption and decryption operations grows with the key size of the cryptosystem. Observe that the encryption cost is proportional to the key size, whereas the key size has minimal impact on the decryption cost, as it only involves modular multiplication and multiplicative inverse calculations.

Further, Figure 5(b) shows the execution time of encryption and decryption as a function of the number of workers $n$, where we fix the key size = 256 bits and $\rho = 0.5$. Clearly, the encryption cost is independent of the number of workers, while the decryption cost grows mildly with $n$. In both cases, the cryptographic operations incur fairly limited system overhead, compared with training the local models.

b) $\rho$-Visibility and Local Differential Privacy: Recall that $\infty$MDL integrates $\rho$-visibility with the local differential privacy (LDP) mechanism. In this set of experiments, we examine their joint impact on the utility of the global model. Specifically, we measure the classification accuracy of the trained models as the settings of LDP and $\rho$-visibility vary. For LDP, we consider the per-parameter privacy budget $\epsilon = 0.01, 0.1, 1$, with smaller budget implying stronger protection; for $\rho$-visibility, we consider the setting of $\rho = 0.125, 0.5$ with larger $\rho$ meaning synchronization among more workers.

As shown in Figure 6, the large accuracy gap between $\epsilon = 0.01$ and 1 ($\rho = 0.125$) indicates the detrimental effect of the LDP mechanism on the model utility. For instance, in the case of CNN-SVHN, the training under $\epsilon = 0.1, 0.01$ even fails to converge. Yet, this negative impact is greatly mitigated by increasing $\rho$. For example, in the case of MLP-MNIST, the accuracy improves by 19.5% as we increase $\rho = 0.125$ to 0.5 under $\epsilon = 1$. Interestingly, this improvement is even more evident in the case of CNN-SVHN (333.0% accuracy boost). This phenomenon is explained by that as more workers aggregate their local gradients, due to the symmetric distribution used in sampling random noise, a larger part of injected noise cancels out, leading to more reliable local gradient estimates. More complicated DNN models (e.g., CNN) tend to be more sensitive to the reliability of such estimates.

Thus, we empirically show that $\rho$-visibility and LDP can be integrated in synergy and existing DP-based PEMs can be enhanced by $\rho$-visibility to achieve better utility.

c) Model Utility and Training Efficiency: Next we study the intricate tradeoff between model utility and training efficiency. We fix the LDP per-parameter budget as $\epsilon = 0.5$.

Figure 7(a) depicts the classification accuracy of $\infty$MDL$^c$ converges under varying settings of $\rho$ (from 0.125 to 1). It is noticed that the model utility (accuracy) of $\infty$MDL$^c$ is, to a large extent, determined by the combination of DNN models and DL tasks. For instance, in the case of MLP-SVHN (i.e., weak model versus hard task), none of the settings of $\rho$ leads to any satisfying accuracy; while in the case of CNN-MLP (i.e., strong model versus easy task), $\infty$MDL$^c$ converges to fairly similar accuracy under varied settings of $\rho$. Meanwhile, observe that the setting of $\rho$ also significantly impacts the model utility under given task-model settings. For example, in the case of CNN-SVHN, the model trained under $\rho = 1$ achieves 91.4% accuracy, in comparison of 81.5% accuracy obtained by that trained under $\rho = 0.125$, which

---

http://deeplearning.net/software/theano/

---

Figure 5: Execution time of encryption and decryption.

Figure 6: Synergy between differential privacy and $\rho$-visibility.
Figure 7: Tradeoff between model utility and training efficiency.

Figure 8: Training efficiency of $\propto$MDL$^c$ and $\propto$MDL$^d$.

We then measure the training efficiency of $\propto$MDL$^c$ under varying $\rho$. As suggested by our analysis, larger $\rho$ implies higher training cost per epoch, as the manager needs to coordinate more workers to commit local gradients. Nevertheless, it is observed in Figure 7(b) that across all DNN model and DL task combinations, the increased cost is not prohibitive. For example, in the case of CNN-SVHN, as $\rho$ increases from 0.125 to 1, the execution time grows about 3 times. However, in large-scale training tasks or low-end system configurations (e.g., limited network bandwidth), a decision has to be made to balance training efficiency and model utility.

d) $\propto$MDL$^c$ versus $\propto$MDL$^d$: We compare the training efficacy of the two variants of $\propto$MDL. Recall our analysis in Section VII that at the expense of slightly higher communication cost, $\propto$MDL$^d$ achieves better training efficiency, especially when different workers possess heterogenous computation capacities.

Figure 9(a) shows the execution time of $\propto$MDL$^c$ and $\propto$MDL$^d$ with $\rho = 0.5$. In both cases of MLP-MNIST and CNN-SVHN, $\propto$MDL$^d$ converges much faster than $\propto$MDL$^c$. For example, in the case of CNN-SVHN, $\propto$MDL$^c$ requires more than 27.0% of training time than $\propto$MDL$^d$ to reach 85%
accuracy, which empirically validates our analysis.

We further consider the setting wherein the workers possess imbalanced computation power. To construct this scenario, we introduce delay in each epoch. The delay time of each worker is drawn randomly from a worker-specific normal distribution. In particular, we set the normal distribution for the i-th worker to be $N(5i, 1)$. As shown in Figure 8(b), $\alpha_{MDL}^d$ demonstrates even more evident advantage than that in Figure 8(a). This implies that $\alpha_{MDL}^d$ is more suitable for scenarios of heterogeneous computation capacities. Astute readers may point out that this may lead to the fairness issue (i.e., workers with weaker computation power may be unable to contribute to MDL at their will). We consider addressing this issue one future research direction.

Also note that although it is often the case that the cost of local training dominates the communication cost, in settings where this premise does not hold (e.g., extremely limited network bandwidth), the superiority of $\alpha_{MDL}^d$ may not hold either. It is up to the MDL operators to choose the right variant of $\alpha_{MDL}$ for given settings.

e) Scalability: In the last set of experiments, we study the scalability of $\alpha_{MDL}$ with respect to the number of workers. Figure 9 shows the training time (250 epochs) of $\alpha_{MDL}$ as the number of workers ($n$) varies from 8 to 16. We consider two settings, $m = \rho \cdot n = 2$ and $m = n/2$.

Observe that with fixed $m$, $n$ has fairly limited impact on the efficiency of $\alpha_{MDL}$. For example, the execution time of $\alpha_{MDL}^c$ increases by 1,076 seconds as $n$ grows from 8 to 16. In comparison, with $\rho$ fixed, $n$ influences the execution efficiency of $\alpha_{MDL}$ more significantly. For example, the running time of $\alpha_{MDL}^c$ increases by 2,318 seconds as $m$ grows from 4 to 8. This indicates that compared with $n$, $m$ has a larger impact on the training efficiency of $\alpha_{MDL}^c$. Also note that compared with $\alpha_{MDL}^c$, $\alpha_{MDL}^d$ is much less sensitive to the setting of $n$ or $m$, thanks to its more dynamic and adaptive nature.

VII. Related Work

DL proves extremely effective at learning nonlinear features and functions from complex data. Besides beating records in image recognition, text classification, and natural language understanding [1], DL outperforms traditional machine learning in healthcare domains, e.g., predicting the effect of mutations in non-coding DNA [25]. The training data in such applications is often highly sensitive, thus requiring incorporative effective privacy enhancing mechanisms (PEMs). Our goal is to protect the input privacy of participating parties in MDL; thus PEMs designed for protecting model privacy [26], [27] and output privacy [28] well complement this work.

Techniques using secure multiparty computation (SMC) help protect input privacy of multiple parties when they collaboratively train machine learning models on their proprietary data. SMC has been applied to design algorithm-specific PEMs, e.g., linear regression functions [12], association rule [11] and Naive Bayes classifiers [29]. However, most of carefully engineered SMC protocols incur non-trivial performance over-