DeepChain: Auditable and Privacy-Preserving Deep Learning with Blockchain-based Incentive

Jia-Si Weng, Jian Weng, *Member, IEEE,* Jilian Zhang, Ming Li, Yue Zhang, Weiqi Luo

Abstract—Deep learning technology has achieved the high-accuracy of state-of-the-art algorithms in a variety of AI tasks. Its popularity has drawn security researchers' attention to the topic of privacy-preserving deep learning, in which neither training data nor model is expected to be exposed. Recently, federated learning becomes promising for the development of deep learning where multi-parties upload local gradients and a server updates parameters with collected gradients, the privacy issues of which have been discussed widely. In this paper, we explore additional security issues in this case, not merely the privacy. First, we consider that the general assumption of honest-but-curious server is problematic, and the malicious server may break privacy. Second, the malicious server or participants may damage the correctness of training, such as incorrect gradient collecting or parameter updating. Third, we discover that federated learning lacks an effective incentive mechanism for distrustful participants due to privacy and financial considerations. To address the aforementioned issues, we introduce a value-driven incentive mechanism based on Blockchain. Adapted to this incentive setting, we migrate the malicious threats from server and participants, and guarantee the privacy and auditability. Thus, we propose to present DeepChain which gives mistrustful parties incentives to participate in privacy-preserving learning, share gradients and update parameters correctly, and eventually accomplish iterative learning with a win-win result. At last, we give an implementation prototype by integrating deep learning module with a blockchain development platform (Corda V3.0). We evaluate it in terms of encryption performance and training accuracy, which demonstrates the feasibility of DeepChain.

Index Terms—Deep learning, Privacy-preserving training, Blockchain, Incentive

1 INTRODUCTION

Recent advances in deep learning based on artificial neural networks have witnessed unprecedented accuracy in various tasks, e.g., speech recognition [1], image recognition [2], drug discovery [3] and gene analysis for cancer research [4], [5]. In order to achieve even higher accuracy, huge amount of data must be fed to deep learning models, incurring excessively high computational overhead [6], [7]. This problem, however, can be solved by employing distributed deep learning technique that has been investigated extensively in recent years. Unfortunately, privacy issue worsens in the context of distributed deep learning, as compared to conventional standalone deep learning scenario.

Privacy-preserving deep learning thus arises to deal with privacy concerns in deep learning, and various models have been around in the past few years [8], [9], [10], [11], [12], [13], [14], [15], [16]. Among these existing work, *federated learning* is the widely adopted system context. Federated learning, also known as *collaborative learning*, *distributed learning*, is essentially the combination of deep learning and distributed computation, where there is a server, called parameter server, maintaining a deep learning model to train and multiple parties that take part in the distributed training process. First, the training data is partitioned and stored at each of the parties. Then, each party trains a deep learning model (the same one as maintained at the parameter server) on her local data individually, and uploads intermediate gradients to the parameter server. Upon receipt of the gradients from all the parties, the parameter server aggregates those gradients and updates the learning model parameters accordingly, after which each of the parties downloads the updated parameters from the server and continues to train her model on the same local data again with the downloaded parameters. This training process repeats until the training errors are smaller than pre-specified thresholds.

This federated learning framework, however, cannot protect the privacy of the training data, even the training data is divided and stored separately. For example, some researchers show that the intermediate gradients can be used to infer important information about the training data [17], [18]. Shokri *et. al* [11] applied differential privacy technique by adding noises in the gradients to upload, achieving a trade-off between data privacy and training accuracy. Hitaj *et. al* [19] pointed out that Shokri's work failed to protect data privacy and demonstrated that a curious parameter server can learn private data through GAN (Generative Adversarial Network) learning. Orekondy *et. al* [20] exploited the intermediate gradients to launch linkability attack on training data, since the gradients contain sufficient data features.

Phong *et. al* [16] proposed to use homomorphic encryption technique to protect training data privacy from curious parameter server. The drawback of their scheme is that they assumed the collaborative participants are honest but not curious, hence their scheme may fail in scenario where some participants are curious. To prevent curious participants, Bonawitz *et. al* [14] employed a secret sharing and symmetric encryption mechanism to ensure confiden-

- *J. S. Weng, J. Weng, J. L. Zhang, M. Li, Y. Zhang and W. Q. Luo are with the College of Information Science and Technology in Jinan University, and Guangdong/Guangzhou Key Laboratory of Data Security and Privacy Preserving, Guangzhou 510632, China.*
E-mail addresses: wengjiasi@gmail.com (J. S. Weng), cryptjweng@gmail.com (J. Weng), jilian.z.2007@smu.edu.sg(J. L. Zhang), limjnu@gmail.com (M. Li), zyueinfosec@gmail.com (Y. Zhang), lwq@jnu.edu.cn (W. Q. Luo).
Jian Weng is the corresponding author.

tiality of the gradients of participants. They assumed that (1) participants and parameter server cannot collude at all, and (2) the aggregated gradients in plain text reveal nothing about the participants' local data. The second assumption, unfortunately, is no longer valid since membership inference attack on aggregated location data is now available [21].

Despite extensive research is underway on distributed deep learning, there are two serious problems that receive less attention so far. The first one is that existing work generally considered privacy threats from curious parameter server, neglecting the fact that there exist other security threats from dishonest behaviors in gradient collecting and parameter update that may disrupt the collaborative training process. For example, the parameter server may drop gradients of some parties deliberately, or wrongly update model parameters on purpose. Recently, Bagdasaryan *et. al* [22] demonstrated the existence of this problem that dishonest parties can poison the collaborative model by replacing the updating model with its exquisitely designed one. Therefore, it is crucial for distributed deep learning framework to guarantee not only confidentiality of gradients, but auditability of the correctness of gradient collecting and parameter update.

The second problem is that in existing schemes those parties are assumed to have enough local data for training and are willing to cooperate in the first place, which are not always true in real applications. For example, in healthcare applications, companies or research institutes are usually facing the difficulty in collecting enough personal medical data, due to privacy regulations such as HIPAA [23] and people's unwillingness to share. As a consequence, lack of training data will result in poor deep learning models in general [24]. On the other hand, in business applications some companies may be reluctant to participate in collaborative training, because they are very concerned about possible disclosure of their valuable data during distributed training [11]. Obviously, it is vital to ensure data privacy and bring in some incentive mechanism for distributed deep learning, so that more parties can actively involved in collaborating training.

In this paper, we propose DeepChain, a secure and decentralized framework based on Blockchain-based incentive mechanism and cryptographic primitives for privacy-preserving distributed deep learning, which can provide data confidentiality, computation auditability, and incentives for parties to participate in collaborative training. The system models of traditional distributed deep learning and our DeepChain are given in Fig. 1. Specifically, DeepChain can securely aggregate local intermediate gradients from untrusted parties through launching transactions, while local training and parameter update are performed by workers (an entity in DeepChain that will be defined shortly) who are incented to process the transactions. Through transaction processing and incentive mechanism, DeepChain achieves collaborative training. Meanwhile, by using cryptographic techniques we ensure data confidentiality and auditability of the collaborative training process as well. To summarize, in this paper we made the following contributions:

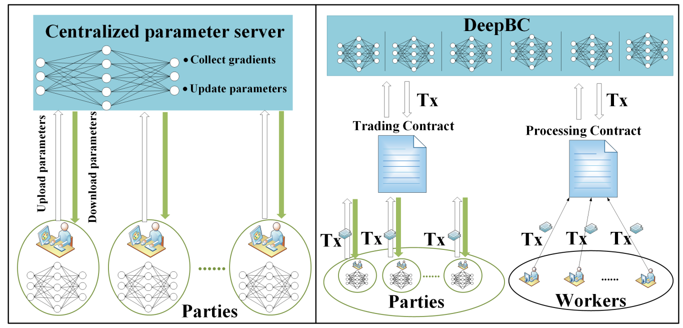

Fig. 1. The left corresponds to traditional distributed deep training framework, while the right is our DeepChain. Here, Trading Contract and Processing Contract are smart contract in DeepChain, together guiding the secure training process, while Tx refers to transaction.

- We propose DeepChain, a collaborative training framework with an incentive mechanism that encourages parties to jointly participate in deep learning model training and share the obtained local gradients.
- DeepChain preserves the privacy of local gradients and guarantees auditability of the training process. By employing incentive mechanism and transactions, participants are pushed to behave honestly, particularly in gradient collecting and parameter update, thus maintaining fairness during collaboration training.
- We implement DeepChain prototype and evaluate its performance in terms of encryption efficiency and training accuracy. We believe that DeepChain can benefit AI and machine learning communities, for example, it can audit collaborative training process and the trained model, which represents the learned knowledge. Making the best use of this learned knowledge by combining transfer learning technique can improve both the learning efficiency and accuracy.

The rest of the paper is organized as follows. In Section 2, we give a brief introduction of Blockchain and deep learning model training. Then, we describe the threat model and security requirements in Section 3. In Section 4, we present our DeepChain, a framework for auditable and privacy-preserving deep learning, and analyze security properties of DeepChain in Section 5. We give implementation details of DeepChain in Section 6, and conduct extensive experiments to evaluate its performance. Finally, we conclude the paper in Section 7.

2 Background

Our work is closely related to Blockchain and deep learning training, and we give background knowledge in this section.

2.1 Blockchain technology

Blockchain was first technology has arisen a surge of interests both in the research community and industry [25]. It becomes an emerging technology as a decentralized, immutable, sharing and time-order ledger. Transactions are stored into blocks containing timestamps and references (i.e., the hash of a previous block) which are maintained as a chain. In Blockchain, transactions are created by pseudonymous participants and competitively collected to build a new block by an entity called *worker*. The worker who builds a new and valid block can gain amount of rewards so that the chain is continuously lengthened by competitive workers. That presents the incentive mechanism in the Blockchain setting. In addition, pro-developing Blockchain technologies introducing smart contract support Turing-complete programmability, such as Ethereum and Hyperledger. On the other hand, a series of works on transaction privacy are popular by applying cryptographic tools into Blockchain, such as Zerocash [26], Zerocoin [27] and Hawk [28]. Therefore, Blockchain technology's incentive feature and its pro-developing technologies inspire us to solve our scenario issues, such as the absence of incentive function and collaboration fairness.

2.2 Deep learning model and training

A typical deep learning model consists of three layers, namely input layer, hidden layer and output layer. A deep learning model can contain multiple hidden layers, where the number of layers is called *depth* of the model. Each hidden layer can have certain amount of neurons, and neurons at different layers can learn hierarchical features of the input training data, which represent different levels of abstraction. Each neuron has multiple inputs and a single output. Generally, the output of neuron i at layer $l-1$ connects to the input of each neuron at layer l. For the connection between two neurons, there is a weight assigned to it. For example, $w_{i,j}$ is the weight associated to the connection between neuron i at layer $l-1$ and neuron j at layer l. Each neuron i also has a bias b_i. Collectively, weights and bias are called *model parameters*, which need to be learned during the training.

Back-Propagation (BP) [29]is the most popular learning method for deep learning, which consists of feed forward step and back-propagation step. Specifically, in feed forward step, the outputs at each layer are calculated based on parameters at previous layer and current layer, respectively.

A key component in deep network training is called *activation*, which is the output of each neuron. Activation is used to learn non-linear features of inputs via a function $Act(\cdot)$. For computing the output value of a neuron i in layer l, $Act(\cdot)$ takes all inputs n of i from layer $l-1$ as input. Additionally, we assume weights $w_{j,i}$ link to the connections between neurons j in layer $l-1$ and neurons i in layer l and b_i links to the bias of neuron i. Then, the value of neuron i in layer l is calculated as $Act_i(l) = Act_i(\Sigma_{j=1}^{n}(w_{j,i} * Act_j(l-1)) + b_i)$. On the other hand, the second step is back-propagation algorithm by using gradient descent. It is to shrink the error E_{total} which are the gaps between model output values V_{output} and target values V_{target}. Assume that there are n output units in the output layer. Then, $E_{total} = \Sigma_{i=1}^{n} 1/2(V_{target_i} - V_{output_i})^2$ is computed. With the error E_{total}, weights $w_{j,i}$ can be updated via $w_{j,i} = w_{j,i} - \eta * \frac{\partial E_{total}}{\partial w_{j,i}}$ so that its gradient decreases. η means the learning rate and $\frac{\partial E_{total}}{\partial w_{j,i}}$ is the partial derivative of E_{total} with respect to $w_{j,i}$. The learning procedure is repeated until the pre-set iteration to train is reached.

When training a rather complex and multi-layer deep learning model, the aforementioned training procedure needs high computation-consuming and time-cost. In order to migrate this problem, distributed deep learning training has been widely discussed, and most of developed excellent systems and architectures exhibit attractive performance, such as DistBelief [30], Torch [31], DeepImage [32] and Purine [33]. There are two approaches for distributed training: model parallelism and data parallelism. The former partitions a total model while the latter partitions the whole training dataset on multiple machines. Our work focuses on the latter one where multiple machines maintain the copy of the training model and process different data subsets being partitioned. These machines share the common parameters of the training model, by uploading and downloading parameters, on a centralized parameter server. Then, multiple machines upload their local training gradients, with which the commonly maintaining model is updated by using SGD. They download updated parameters from the parameter server and continue to train the local model. With iteratively training, those machines at the end together gain the trained model. Recently, a series of works on distributed deep learning training are continue to be proposed [34], [35], [36], [37], [38], which showed us the feasibility to research on collaborative training a deep learning model.

3 THREATS AND SECURITY GOALS

In this section, we discuss threats to collaborative learning, and security goals that BeepBC can achieve to tackle those threats.

Threat 1: Disclosure of local data and model. Although in distributed deep training each party only uploads her local gradients to the parameter server, still adversaries can infer through those gradients important information about the party's local data by initiating an inference attack or membership attack [18]. On the other hand, based on the gradients adversaries may also launch parameter inferring attack to obtain sensitive information of the model [19].

Security Goal: Confidentiality of local gradients. Assume that participants do not expose their own data and at least t participants are honest (i.e., no more than t participants colluded to disclose parameters). Then each party's local gradients cannot be exposed to anyone else, unless at least t participants collude. In addition, if in any circumstance participants do not disclose the downloaded parameters from the collaborative model, then adversaries could not gain any information about the parameters. To achieve this goal, in DeepChain each participant individually encrypts and then uploads gradients obtained from her local model. All gradients are used to update parameters of the collaborative model encrypted collaboratively by all participants, who then obtain updated parameters via collaborative decryption in each iteration (collaborative decryption refers to at least t participants provide their secret shares to decrypt a cipher).

Threat 2: Participants with inappropriate behaviors. Consider a situation that participants may have malicious behaviors during collaborative training. They may choose their inputs at will and thus generate incorrect gradients, aiming to mislead the collaborative training process. As a consequence, when updating parameters of collaborative model using the uploaded gradients, it is inevitable that we will get erroneous results. On the other hand, in collaborative decryption phase dishonest participants may give a problematic decryption share and they may be selfish, aborting local training process early to save their cost for training. In addition, dishonest participants may delay trading or terminate a contract for her own benefit. All these malicious behaviors may fail the collaborative training task.

Security Goal 1: Auditability of gradient collecting and parameter update. In DeepChain, assume that majority of the participants and more than $\frac{2}{3}$ of the workers are honest in gradient collecting and parameter update, respectively. During gradient collecting, participants' transactions contain encrypted gradients and correctness proofs, allowing the third party to audit whether a participant gives a correctly encrypted construction of gradients. For parameter update, on the other hand, workers claim computation results through transactions that will be recorded in DeepChain. These transactions are auditable as well, and computation results are guaranteed to be correct only if $\frac{2}{3}$ workers are honest. After parameters are updated, participants download and collaboratively decrypt the parameters by providing their decryption shares and corresponding proofs for correctness verification. Again, anyone third party can audit whether the decryption shares are correct or not.

Security Goal 2: Fairness guarantee of participants. DeepChain provides fairness for participants through timeout-checking and monetary penalty mechanism. Specifically, for each function with smart contracts DeepChain defines a time point for it. At the time point after function execution, results of the function are verified. If the verification failed, it means that (1) there exist participants not being punctual by the time point, and (2) some participants may incorrectly execute the function. For either of the two cases, DeepChain applies the monetary penalty mechanism, revoking the pre-frozen deposit of dishonest participants and re-allocating it to the honest participants. Therefore, fairness can be achieved, because penalty will never be imposed on honest participants behaved punctually and correctly, and they will be compensated if there exist dishonest participants.

4 The DeepChain Model

In this section, we present DeepChain, a secure and decentralized framework for privacy-preserving deep learning.

4.1 System overview

Before introducing DeepChain, we give definitions of related concepts and terms used in DeepChain.

• **Party:** In DeepChain, a party is the same entity as defined in traditional distributed deep learning model, who has similar needs but unable to perform the whole training task alone due to resource constraints such as insufficient computational power or limited data.

• **Trading:** When a party got her local gradients, she sends out the gradients through a smart contract called *trading contract* to DeepChain. This process is called *trading*. Those contracts can be downloaded to process by *worker* (an entity in DeepChain that will be defined shortly).

• **Cooperative group:** A cooperative group is a set of parties who have a same deep learning model to train.

• **Local model training:** Each party trains her local model independently, and at the end of a local iteration the party generates a contract to trade by attaching her local gradients to the contract.

• **Collaborative model training:** Parties of a cooperative group train a deep learning model collaboratively. Specifically, after deciding a same deep learning model and parameter initialization, the model is trained in an iterative manner. In each iteration, all parties trades their gradients, and workers download the contracts to process the gradients. The processed gradients then send out by workers through smart contract called *processing contract*. The correct processed gradients are picked out and used to update parameters of the common model on DeepChain. Parties download the updated parameters from the common model and update their local models accordingly. After that the parties begin next training iteration.

• **Worker:** Similar to *miners* in BitCoin, workers are incented to process transactions that contain training weights for collaborative model update. Workers compete to work on a block, and the first one finishes the job is a leader. The leader will gain block rewards which can be consumed in the future. Maybe he can exchange them with trained models in DeepChain for accomplishing his AI task.

• **Iteration:** Deep learning model training consists of multiple steps called *iterations*, where at the end of each iteration all the weights of neurons of the model are updated once.

• **Round:** In DeepChain, a *round* refers to the process of the creation of a new block.

• **DeepCoin:** DeepCoin, denoted as $\$Coin$, is a kind of asset on DeepChain. In particular, for each newly generated block DeepChain will generate certain amount of $\$Coin$ as rewards. Participants in DeepChain consist of parties and workers, where the former gain $\$Coin$ for their contributions to local model training, and the latter are rewarded with $\$Coin$ for helping parties update training models. Meanwhile, a well-trained model will cost $\$Coin$ for those who have no capability to train the model by themselves and want to use the model. This setting is reasonable because recent work on model-based pricing for machine learning has found applications in some scenarios [39], [40]. We define a *validity value* for $\$Coin$, which essentially is the time interval of a round. Validity value is related to consensus mechanism in DeepChain, and we will discuss it in detail in 4.2.5.

DeepChain combines together Blockchain techniques and cryptographic primitives to achieve secure, distributed, and privacy-preserving deep learning. Suppose there are N parties P_j, $j = 1, ..., N$, and they agree on some predefined information such as a concrete common model and initial parameters for the common model. The information is attached to a transaction Tx^0_{com} (com represents 'common') signed by all parties. Assume an address corresponding to transaction Tx^0_{com} is pk_{it_0}, where it_0 is the initial iteration. At the end of iteration i, the updated model in Tx^i_{com} is attached to a new address pk_{it_i}. All addresses are known to the parties.

Intermediate gradients from party P_j are enveloped in transaction $Tx^i_{P_j}$, and all those transactions are collected by a *trading contract* at iteration i. Note that intermediate gradients are local weights $C_{P_j}(\triangle \mathbf{W}_{i,j})$, where C is a cipher used by party P_j to encrypt the weights. When all transactions $\{Tx^i_{P_j}\}$ at iteration i have been collected, trading contract uploads them to DeepChain. After that, workers download those transactions $\{Tx^i_{P_j}\}$ to process via *processing contract*. Specifically, workers update the weights by computing $C(\mathbf{W}_{i+1}) = \frac{1}{N} \cdot C(\mathbf{W}_i) \cdot \prod_{j=1}^{N} C_{P_j}(\triangle \mathbf{W}_{i,j})$, where $C(\mathbf{W}_i)$ is the weight at iteration i in Tx^i_{com}, and $C(\mathbf{W}_{i+1})$ is the updated weights that will be attached to Tx^{i+1}_{com} for updating the local models in next iteration $i+1$.

4.2 Components of DeepChain

Our concrete design includes five building blocks: DeepBC bootstrapping, incentive mechanism, party assert display, cooperative training and consensus protocol.

4.2.1 DeepBC bootstrapping

DeepBC bootstrapping accomplishes two things: (1) DeepCoin distribution and (2) genesis block generation which are innate elements for running DeepBC. On one hand, we assume that users have registered on DeepBC and each user uses one of pks as the address corresponding to a DeepCoin unit he launches a transaction. DeepCoin distribution realizes that each user on DeepBC is allocated amounts of DeepCoins. We assume that the quantity of the allocations are equal. On the other hand, the genesis block contains initial transactions referring to DeepCoin possession statements after the step of DeepCoin distribution. Assume that the round begins with 0 and generates the genesis block. When the genesis block is generated, a random seed $seed_0$ also is public. $seed_0$ is randomly chosen by initial users via using distributed random number generation. We note that $seed_0$ is the base random seed for DeepBC. Particularly, $seed_0$ is one of components to choose the random seed $seed_1$ in the round with the index 1 and the rest of rounds can be done in the same manner. The seeds are crucial to guarantee the randomness to select a new leader who creates a new block. This follows the idea of cryptographic sortition from Algorand [41], [42]. We will introduce it in section $4.2.5$ for the stable running of DeepBC.

4.2.2 Incentive mechanism

An incentive is the motivation for a party to act. Designing an incentive mechanism is to produce value and leads to collective benefit. By DeepBC's incentive mechanism, individual parties are value-driven to exchange gradients with others for obtaining collaborative models which are well-trained. On one hand, it promotes the cooperation of parties who desire but fail to gain a well-trained deep learning model solving AI tasks due to data sparseness. On the other hand, it encourages cooperative parties to honestly trade locally training gradients, and makes workers honestly process parties' transactions where they update parameters of collaborative models with traded gradients.

This incentive mechanism is value-driven and introduces monetary rewards and penalties for participants' performances. For giving a better understanding, we suppose a scenario where two individual parties possess small quantities of data which cannot allow individuals to train an high-accuracy model; combining their data makes it possible for them to achieve the high-accuracy training. DeepBC enables two parties to combine their data for collaborative training via launching transactions. Data possessed by two parties are assumed in the unequal distribution. Parties launch transactions and pay a few transaction fees, the amount of which are related to the quantity distribution of individual data; the more data an individual owns, the less fees he should pay. Suppose that two parties constitute a group and agree on the amount of fees for collaborative training; they also promise the fees each party should pay according to the quantity distribution. Assume that each party needs to make ten thousand transactions for accomplishing collaborative training; the total fees paid by those ten thousand transactions should be equal to the value he promised. Note that transactions containing iterative gradients would be iteratively processed by workers who compete for a leader to earn rewards by successfully creating a new block. In addition, transaction making and processing are verifiable. If an invalid transaction made by some party is caught, the party would be punished. Meanwhile, a leader who incorrectly processes transactions also is punished, which reduces his reputation. At the end, the accomplished model creates value for these two parties solving their AI tasks and serving others via paying.

To give a formalized description for the incentive mechanism, we first introduce two properties, compatibility and liveness of the incentive mechanism for parties and workers, which demonstrates collaborative value on DeepBC. Then, we further explain it that parties and workers have incentives to behave honestly. Assume that we guarantee data privacy and the security of the consensus protocol which are to be introduced. We also assume that the value v_c of the collaboratively well-trained model is higher than the value v_i of individually trained model in terms of the quantity of training data.

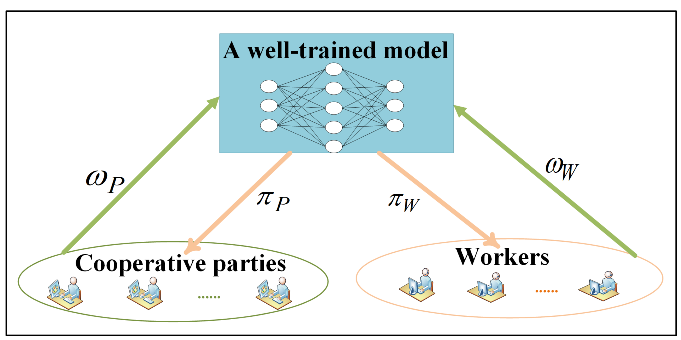

Fig. 2. The incentive mechanism of DeepBC. ω_P and ω_W represent the contributions of a party and a worker for maintaining v_c, respectively; π_P and π_W represent their payoffs, respectively. In section 4.2.2, we illustrate that only both sides make most contributions to a well-trained model $Max(\omega_P) \bigwedge Max(\omega_W)$, they can gain the highest total payoff $Max(\pi_P, \pi_W)$.

First, we say it has compatibility if each party can obtain the best result just by performing according to their true propensities. Meanwhile, it has liveness that v_c is maintained, only if each party trends to transform v_i, and each worker has incentives at v_c. Both honest parties and workers have the same common propensity to gain well-trained models, represented as v_c. For a party, he should iteratively perform to trade gradients for v_c, which is defined as $cost(v_i)$. If the party desires to obtain a collaboratively well-trained model, he needs to accomplish his entire participancy. For a worker, he should process transactions for v_c, to earn rewards with probability and gain reputation. Money he earned enables him to pay AI services on DeepBC. Note that the probability is said that a worker has a probability to gain rewards according to the quantity of money and reputation he has earned, i.e., the larger the quantity is, the higher probability would be. As a result, a worker in order to gain rewards with the higher probability, has incentives to maintain v_c. We further use ω_P and ω_W to represent the contributions of a party and a worker for maintaining v_c, respectively; using π_P and π_W to represent their payoffs from maintaining v_c, respectively. For an individual party, the more he contributes ω_P, the more he gains π_P and that rule also holds for a worker. Then, within a collaborative training, both sides play incentively to a well-trained model $Max(\omega_P) \bigwedge Max(\omega_W)$, the total payoff is highest gained by them $Max(\pi_P, \pi_W)$. If any participant can not perform well $(\omega_P = 0) \bigvee (\omega_W = 0)$, nothing would be got that is $(\pi_P = 0) \bigwedge (\pi_W = 0)$ where $\bigwedge$ means $'and'$ and $\bigvee$ means $'or'$.

Payoff=

$$\begin{cases} Max(\pi_P \bigwedge \pi_W) & \textbf{If} \quad Max(\omega_P) \bigwedge Max(\omega_W) \\ (\pi_P = 0) \bigwedge (\pi_W = 0) & \textbf{If} \quad (\omega_P = 0) \bigvee (\omega_W = 0) \end{cases}$$

Second, based on the aforementioned description, we show each party and worker are value-driven to behave honestly in each iteration so that they can obtain the highest payoff, in which the theory derives from the work [43]. We formalize it as $Value(1) = \pi_P - \omega_P(1)$ for a party. Assume that the method is correct with the probability $Pr_v(P)$ to verify a party's malicious behavior is malicious and $Pr_v(W)$ is similar to a worker; the mechanism to launch a penalty is assumed to be designed securely. Note that a party's contribution is defined as ω_P. For party, we say that ω_P is honestly provided with the probability $Pr_c(P)$. Then, $\omega_P(Pr_c(P))$ is used to represent his true performance. In addition, we get the probability $Pr_v(P)*(1-Pr_c(P))$ that a dishonest party would be caught. Once the dishonest party is caught, he is punished by forfeiting his deposit, the loss of which is defined as f_P. Thus, the returned value according to the party's true behavior can be represented as

$$Value(Pr_c(P)) = \pi_P*(1-Pr_{vc}(P))-f_P*Pr_{vc}(P)-\omega_P(Pr_c(P))$$

where $Pr_{vc}(P) = Pr_v(P) * (1 - Pr_c(P))$. We expect that the value is max only when the party behaves honestly $Pr_c(P) = 1$ and then $Value(1) = \pi_P - \omega_P(1)$ can hold. This indicates the significance of the incentive mechanism. We can achieve this expectation by setting the values of $Pr_v(P)$, π_P, and f_P as following.

Theorem 1. If $f_P/\pi_P > (1 - Pr_{vc}(P))/Pr_{vc}(P)$ where $Pr_{vc}(P) = Pr_v(P) * (1 - \theta)$ is set, then a party will be honest at least with the probability θ.

TABLE 1
Notations and implications

Notations	Implications
pk_P^{psu}	a pseudo-generated public key of party P
sk_P	a secret key of the party P
q	a randomly selected big prime
G_1	cyclic multiplicative cyclic groups of prime order q
G_2	cyclic multiplicative cyclic groups of prime order q
g	a generator of group G_1
Z_q^*	{1,2, ..., q-1}
e	a bilinear map e: $G_1 \times G_1 \rightarrow G_2$
H_1	a collision-resistant hash function mapping any string into an element in Z_q^*
H_2	a collision-resistant hash function mapping any string into an element in G_1
$C()$	a cipher generated by Paillier.Encrypt algorithm
$Enc()$	the encryption by individual parties

Proof. It can be significant by proving that for any $Pr_c^{'}(P) < \theta$, $Value(Pr_c^{'}(P))$ is lower than $Value(\theta)$. Without the loss of generality, we prove for any $Pr_c^{'}(P) < \theta$, $Value(Pr_c^{'}(P))$ is lower than 0. That is $Value(Pr_c^{'}(P)) = \pi_P * (1 - Pr_{vc}^{'}(P)) - f_P * Pr_{vc}^{'}(P) - \omega_P(Pr_c^{'}(P))$ is lower than 0. When we set $f_P/\pi_P > 1/Pr_{vc}^{'}(P) - 1$, the result $\pi_P*(1-Pr_{vc}^{'}(P)) - f_P*Pr_{vc}^{'}(P)$ is lower than 0. Thus, in this case, $Value(Pr_c^{'}(P))$ is lower than 0 that holds.

For a worker, the incentive analysis is similar to the analysis for a party, expect that his payoff has probability to gain. We set this probability is Pr_{leader}. Thus, we should set the relationship of four values Pr_{leader}, $Pr_v(W)$, π_W, and f_W to encourage a worker to be honest.

Theorem 2. If $f_W/\pi_W * Pr_{leader} > (1 - Pr_{vc}(W))/Pr_{vc}(W)$ where $Pr_{vc}(W) = Pr_v(W) * (1 - \epsilon)$ is set, then a worker will be honest at least with the probability ϵ.

Proof. The proof is similar to the proof for **Theorem** 2, so it is omitted.

4.2.3 User assert display

We call a user's data as his assert, and the assert's value is recognized by DeepBC, which are essential for creating collaborative value. A user needs to display his assert on DeepBC first, then finds cooperators and accomplishes his AI tasks. Assert display only claims which AI tasks his data is related to while preserves data privacy. Formally, an assert display is represented via a transaction including four parts. Note that a transaction is launched by a pseudo public key address. The pseudo public key address is generated by the user according to his wishes. The process to generate pseudo public keys is shown as follows.

$$pk_P^{psu} \in \{g_1^{sk_P}, g_2^{sk_P}, ..., g_n^{sk_P}\}$$

It indicates the party P wishes to have n public keys (n is an integer). P selects a secret key $sk_P \in Z_q^*$ and generates a series of public keys $g_i^{sk_P} \in G_1$ where g_i means that a random element $r_i \in Z_q^*$ pow g, as well as i is in $[1, n]$. q and g are system parameters on DeepBC while r_i is secretly selected by individual parties. Necessarily, we list related cryptographic notations in TABLE 1. Thus, suppose that the assert display of the party P_1 can be represented via the following transaction with the address $pk_{P_1}^{psu}$.

$$pk_{P_1}^{psu} \rightarrow \Big\{ \Big(pk_{data\_P_1} = g^{H_1(data\_P_1)}, \\ \sigma_{j\_P_1} = (H_2(j) \cdot g^{H_1(data_{j\_P_1})})^{H_1(data\_P_1)} \Big), \\ \text{"Keywords"} \Big\}$$

The first part $pk_{data\_P_1}$ without leaking the value of $H(data\_P_1)$ is regarded as the assert proof that P_1 indeed possesses data $H(data\_P_1)$. Particularly, $\sigma_{j\_P_1}$ contains l components since $data\_P_1$ is divided into l blocks, each of which is $data_{j\_P_1}$ where j is in $[1, l]$. The second part announces "Keywords" as the description for the raw data $data\_P_1$, which helps a user to find cooperators with the similar AI tasks. When implemented, "Keywords" are formed with the `JSON` style which include 4 fields: data size, data format, data topic and data description. Then, P_1 submits his assert transaction as $Tran_{P_1}$. We assume that displayed data in the first time on DeepBC are authentic which is reasonable in the setting of Blockchain.

4.2.4 Collaborative training

Due to the phase of user assert display, parties who have similar AI tasks can constitute a group for collaborative training the following four steps.

• **Collaborative group establishment.** Note that keywords related to AI tasks are displayed in the prior phase. Parties establish a collaborative group $Group$ according to similar "Keywords". They may get more information about "Keywords" by off-line interactions and further confirm their collaboration, the details of which are omitted in the paper. In this step, pre-cooperative parties can audit cooperators' data to ensure the authenticity of data possession. The auditing process can be referred to the full paper[?]. N parties P_1, P_2, ..., P_N constitute $Group$ with pseudonymity, i.e, pseudo public keys $pk_{P_1}^{psu}, pk_{P_2}^{psu}, ..., pk_{P_N}^{psu}$, and their secret keys $sk_{P_1}, sk_{P_2}, ..., sk_{P_N}$ are privately possessed, respectively. Since parties may launch transactions by using different pseudo public keys, the transactions signed by the same secret key sk_{P_i} can be verified that those transactions are from the same cooperative member.

• **Collaborative information commitment.** After $Group$ establishment, parties agree on their collaborative information for securely training a common deep learning model as follows. In this step, we assume a trusted component only for the setup phase in Threshold Paillier algorithm, and it does not take part in other processes. If there exists no such a trusted component, we can accomplish the setup phase by a distributed way [44].

(1) The number of cooperative parties, N.

(2) The current round index, $round$.

(3) Setup parameters of Threshold Paillier algorithm: $PK_{model} = (n_{model}, g_{model}, \theta = as, V = (v, \{v_i\}_{i\in[1,...,N]}))$ where $g_{model} \in Z^*_{n^2_{model}}$ and $a, s, \theta, v, v_i \in Z^*_{n_{model}}$. $SK_{model} = s$ which is randomly divided into N parts where $s = f(s_1 + ... + s_N)$ (f means a function for secret sharing protocol). Each party owns a part of secure key s_i. v and $\{v_i\}_{i\in[1,...,N]})$ are public verification information, and v_i is in respect to s_i. A threshold $t \in \{N/2, ..., N\}$ is set that more than t parties can together decrypt a cipher. Note that training gradients to be encrypted are vectors with multi-element, such as $\triangle\mathbf{W_{i,j}} = (w^1_{i,j}, ..., w^{\omega}_{i,j})$ where the length of $\triangle\mathbf{W_{i,j}}$ is ω. Note that i represents the iteration index of training and $j \in \{1, ..., N\}$. Considering the problem of the cipher expansion, we encrypt a vector into a cipher instead of multiple ciphers with respected to multiple elements. Then, to choose a ω-length super increasing sequence $\vec{\alpha} = (\alpha_1 = 1, ..., \alpha_\omega)$ that simultaneously meets two conditions. Suppose that each value of $w^1_{i,j}, ..., w^{\omega}_{i,j}$ is not larger than the value d. The two conditions are $\sum_{l=1}^{i-1} \alpha \cdot N \cdot d < \alpha_i (i = 2, ..., \omega)$ and $\sum_{i=1}^{\omega} \alpha_i \cdot N \cdot d < n_{model}$. Then compute $(g^1_{model}, ..., g^{\omega}_{model}) = (g^{\alpha_1}_{model}, ..., g^{\alpha_\omega}_{model})$.

(4) A common deep learning model $model_{com}$ attached to a commonly coordinated address pk^{psu}_{com} is publicly recorded as a transaction $Tran_{com}$ on DeepBC.

$$Tran_{com} = pk^{psu}_{com} \rightarrow model_{com}$$

For $model_{com}$, they agree on which kinds of training networks, which kinds of training algorithms and what configurations of networks (such as the number of network layer, the number of neuron each layer, the size of mini-batch and the times of iteration). Beside these information, they also agree on the initial weights $\mathbf{W}_0$ of $model_{com}$. The weights are protected by applying **Paillier.Encrypt** algorithm $C(\mathbf{W}_0) = g^{\mathbf{W}_0}_{model} \cdot (k_0)^{n_{model}}$ where k_0 is randomly selected from $Z^*_{n_{model}}$. Note that we compute $g^{\mathbf{W}_0}_{model}$ with the help of the chosen super increasing sequence that $g^{\mathbf{W}_0}_{model} = g^{\alpha_1 \cdot w^1_0 + ... + \alpha_\omega \cdot w^\omega_0}_{model}$ so that we generate a cipher for a vector.

(5) A commitment on $SK_{model} = s$ is combined with individual parties' commitments on their individual secret keys s_i.

$$commit_{SK_{model}} = (Enc(s_1||index_r||Sign(s_1||index_r)), \\ ..., Enc(s_N||index_r||Sign(s_N||index_r)))$$

(6) Individual parties' initial weights $\mathbf{W}_{0,j}$ are provided in the encryption form by **Paillier.Encrypt** algorithm. $C(\mathbf{W}_{0,j}) = g^{\mathbf{W}_{0,j}}_{model} \cdot (k_j)^{n_{model}}$, where k_j are randomly selected from $Z^*_{n_{model}}$ by individual parties $P_{j\in\{1,...,N\}}$.

(7) Each cooperative party is required to commit amounts of deposits $\$deposit$ for secure computation. During the cooperation, if a party behaves badly, the deposits would be forfeited and compensated other honest parties. If not, those deposits would be refunded.

(8) Another component $\$Coin$ is committed by each party according to the quantity of data they share. This component can be coordinated off-chain and different parties may commit different amounts of $\$Coin$ with agreement.

We note that two kinds of roles are defined for parties in $Group$, trader and manager, which will be further discussed. All collaborative information are recorded in a transaction $Tran_{com_1}$ being uploaded to DeepBC.

After the aforementioned steps, we next introduce how collaborative training is securely accomplished via two smart contracts, *Trading Contract* and *Processing Contract*. First, parties iteratively trade their gradients to *Trading Contract* executed by the manager who can be selected from traders. The trading gradients are honestly encrypted by each trader and meanwhile the correct proofs of encryption are attached, which indicates two security requirements (confidentiality and auditability). In terms of confidentiality, only if a trader does not disclose his gradients, no one can gain any information. Traders (at most t parties) in addition need to cooperatively decrypt parameters after they are updated in the contract. We assume the manager cannot disclose what he knows, as the work [28] promised. In terms of auditability, each trader sending his encrypted gradients also needs to give a correct proof for it. When cooperatively decrypting, each trader also presents his decryption proof. Those proofs are non-interactively public on DeepBC and auditable for any party. On the other hand, the behaviors of traders and the manager are forced to be authentic and fair by utilizing the timeout-checking and monetary penalty mechanisms. Even if the manager colludes with traders, the outcome of *Trading Contract* cannot be modified [28]. Second, *Processing Contract* is responsible for parameter updating. Workers process transactions by casting up gradients in respect to a group, and send computation results to *Processing Contract*. *Processing Contract* verifies correct computation results and updates model parameters for this group. For accomplishing the whole training, these two contracts are called for multiple times. Concretely, we give more detail descriptions for these two contracts in the following steps.

• **Gradient collecting via Trading Contract.** As shown in **Algorithm 1**, it defines six major functions. With those functions being invoked iteratively, gradient transactions for training $model_{com}$ are securely collected and processed. For the purpose of time-out checking, there declares time points $T_{t_1}, T_{t_2}, T_{t_3}, T_{t_4}, T_{t_5}, T_{t_6}$ following functions, respectively. We stress that the intervals between the time points T_{t_1} and T_{t_6} are declared according to the interval from the time of one iteration i to the time of its next iteration $i+1$, i.e., $|T_{t_6} - T_{t_1}| \leq |T_{i+1} - T_i|$. The time points are set meeting $T_{t_1} < T_{t_2} < T_{t_3} < T_{t_4} < T_{t_5} < T_{t_6}$. At a defined time point, $checkTimeout$ is responsible for checking whether each party behaves honestly or not, before the defined time point. If not, the monetary penalty mechanism performs by forfeiting deposits of the malicious parties, and the failed step is re-executed. With being iteratively invoked, time points are updated, e.g., $T^{'}_{t_1} = T_{t_1} + |T_{i+1} - T_i|$.

Algorithm 1: Trading($Tran^i_{P_1}$,...,$Tran^i_{P_N}$)

receiveGradientTX()
checkTimeout(T_{t1})
updateTime() $\#T^{'}_{t_1} = T_{t_1} + |T_{i+1} - T_i|$
verifyGradientTX()
checkTimeout(T_{t2})
updateTime() $\#T^{'}_{t2} = T_{t2} + |T_{i+1} - T_i|$
uploadGradientTX()#attaching to the address pk^{psu}_{com}
checkTimeout(T_{t3})
updateTime() $\#T^{'}_{t3} = T_{t3} + |T_{i+1} - T_i|$
downloadUpdatedParam()#from the address pk^{psu}_{com}
checkTimeout(T_{t4})
updateTime() $\#T^{'}_{t4} = T_{t4} + |T_{i+1} - T_i|$
decryptUpdatedParam()
checkTimeout(T_{t5})
updateTime() $\#T^{'}_{t5} = T_{t5} + |T_{i+1} - T_i|$
return()
checkTimeout(T_{t6})
updateTime() $\#T^{'}_{t6} = T_{t6} + |T_{i+1} - T_i|$

In particular, in i_{th} iteration, parties launch transactions with encrypted gradients adding publicly auditable proofs for encryption correctness, and send them to $receiveGradientTX()$. Transactions are formulated as follows.

$$Tran^i_{P_j} = \{pk^{psu}_{P_j} : (C(\triangle \mathbf{W}_{i,j}), Proof_{PK_{i,j}}) \rightarrow pk^{psu}_{com}\}$$
$$Proof_{PK_{i,j}} = fsprove_1(\Sigma_{PK}; C(\triangle \mathbf{W}_{i,j}); \triangle \mathbf{W}_{i,j}, k_j; pk^{psu}_{P_j})$$

Then, $verifyGradientTX()$ verifies the correctness of encryption via $fsver_1(\Sigma_{PK}; C(\triangle \mathbf{W}_{i,j}); Proof_{PK_{i,j}}; pk^{psu}_{P_j})$. It verifies whether $C(\triangle \mathbf{W}_{i,j})$ is really an encryption of $\triangle \mathbf{W}_{i,j}$ with the randomness of k_j or not. Here, $pk^{psu}_{P_j}$ is the identity information attached to the proof, which resists the attack of replay proof by malicious parties. Before the time point T_{t_3}, $uploadGradientTX()$ uploads the transactions which are verified to be true. In *Processing Contract*, we will introduce how those transactions are processed making gradients $\sum_{j=1}^{N} \triangle \mathbf{W}_{i,j}$ be contributed to the model $model_{com}$. When model parameters are updated, $downloadUpdatedParam$ undertakes to pull the latest parameters. Suppose the latest iteration at the current moment is i, the cipher of the latest parameters is $C(\mathbf{W}_i)$ from $C(model_{com_i})$ (and simply noted as C_i). $decryptUpdatedParam()$ enables parties to perform individual decryption shares on C_i, combining with a proof of correct decryption.

$$C_{i,j} = C_i^{2\Delta s_j}$$
$$Proof_{CD_{i,j}} = fsprove_2(\Sigma_{CD}; (C_i, C_{i,j}, v, v_j); \Delta s_j; pk^{psu}_{P_j})$$

The proof $Proof_{CD_{i,j}}$ supports to be verified the validity of decryption shares, i.e., $\Delta s_j = log_{C_i^4}(C^2_{i,j}) = log_v(v_j)$ via $fsver_2(\Sigma_{CD}; (C_i, C_{i,j}, v, v_j); Proof_{CD_{i,j}}; pk^{psu}_{P_j})$. If majority of parties $|H| >= N/2$ are honest, then C_i can be correctly recovered by $((\Pi_{j \in H} C^{2\mu_j}_{i,j} - 1)/n_{model})(4\Delta^2\theta)^{-1} mod \quad n_{model}$ where μ_j is Lagrange interpolation coefficient in respect to P_j, and the cleartext is pushed to parties by $return$.

Algorithm 2: Processing()

updateTX()
checkTimeout(T_{t7})
updateTime() $\#T^{'}_{t7} = T_{t7} + T_r$
verifyTX()
checkTimeout(T_{t8})
updateTime() $\#T^{'}_{t8} = T_{t8} + T_r$
appendTX()
checkTimeout(T_{t9})
updateTime() $\#T^{'}_{t9} = T_{t9} + T_r$

• **Parameter updating via Processing Contract.** Suppose that in i_{th} iteration for $model_{com}$, incentive workers competitively execute update operations with all parties' gradients $\triangle \mathbf{W}_{i,j}$ uploaded by *Trading Contract*. Note that to protect the confidentiality of individuals' gradients is one of our goals. Update operations actually are executed on encrypted parameters as follows. $\mathbf{W}_i = \mathbf{W}_{i-1} - 1/N \times \sum_{j=1}^{N} \triangle \mathbf{W}_{i,j}$, it actually is computed in encrypted values as follows.

$$C(\mathbf{W}_i) = C(\mathbf{W}_{i-1}) \cdot 1/N \times (C(-\triangle \mathbf{W}_{i,1}) \cdot C(-\triangle \mathbf{W}_{i,2}) \cdot \text{......} \cdot C(-\triangle \mathbf{W}_{i,N}))$$

According this, workers make transactions including the newly updated parameters and send them to *Processing Contract* ahead of T_{t7}. In the meantime, a leader is randomly chosen from those workers via the consensus protocol on DeepBC. Note that this time the reward for the leader is frozen before verifying his computation work. With $verifyTX$, the correctness of the leader's work is verified by the method that the minority is subordinate to the majority. That is, the result of $C(\mathbf{W}_i)$ given by the leader will be compared with the ones of other competitive workers, and the result is regarded to be correct if the values of the majority are equal to it. If it is incorrect, the leader would be punished according the monetary penalty mechanism, which reduces his reputation on DeepBC, and he gains no reward. Moreover, adapted to DeepBC's consensus protocol (introduced by section 4.2.5), the behavior history influences the probability for him to be a leader on DeepBC. In this case, a leader is re-chosen with his correct computation which is able to be check ahead of T_{t8}. At the end, the leader's block collecting correct transactions with correctly updated parameters is appended to DeepBC. For training, the next iteration $(i+1)_{th}$ begins and $model_{i+1}$ is generated based on $model_i$.

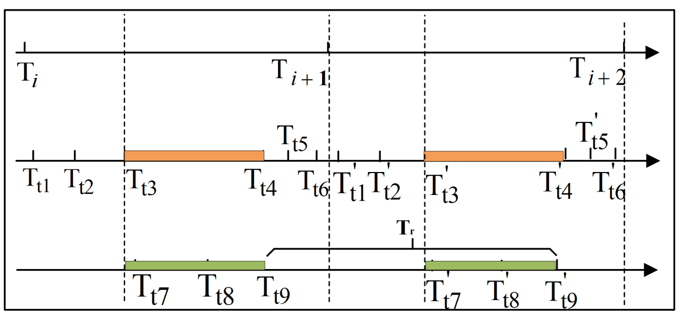

Fig. 3. Configurations on time points. From top to bottom: the timeline of the iterative training, the timeline of trading (in Trading Contract), the timeline of block creation (in Processing Contract).

As shown in *Trading Contract* and *Processing Contract*, the importance of the trusted time lock mechanism is presented. We go back to stress that in *Processing Contract*, time points T_{t7}, T_{t8}, T_{t9} will be updated by $T_{t7}^{'} = T_{t7} + T_r, T_{t8}^{'} = T_{t8} + T_r, T_{t9}^{'} = T_{t9} + T_r$, respectively. T_r means the interval creating a new block within two rounds on DeepBC, as shown in Fig. 3 which depicts the time point configurations involving two contracts. Suppose during i_{th} iteration, defined time points are configured meeting $T_{t_1} < T_{t_2} < T_{t_3} <= T_{t_7} < T_{t_8} < T_{t_9} <= T_{t_4} < T_{t_5} < T_{t_6}$. Meanwhile, the relationship among these three time-lines also shows $Tr \leq |T_{t_6} - T_{t_1}| \leq |T_{i+1} - T_i|$.

In addition to the trusted time lock mechanism, we employ the secure monetary penalty mechanism to present fairness for the procedures of gradient collecting and collaborative decryption. we introduce the formalized methodology proposed by Bentov *et. al* and Kumaresan *et. al* [45], [46] into the **Algorithm 4** F_{ct}^{*} based on *Trading Contract*. In particular, in the process of **Gradient collecting**, fairness is guaranteed twofold: 1) honest collaborative parties must launch gradient transactions which are verified to be correct ahead of the defined time; 2) dishonest parties who launch incorrect transactions or time-out transactions are to be penalized and meanwhile the remained honest are to be compensated. Meanwhile, fairness for collaborative decryption are provided in the process of **Collaborative decryption**, which depicts twofold: 1) a party who gives a correct decryption share at a defined time point never has to pay any penalty; 2) If the adversary successfully decrypt, but a party cannot, then the party should be compensated.

4.2.5 Consensus protocol

Consensus protocol is the essential protocol where all parties make consensus on the same and correct values in the decentralized setting. In the blockchain setting, it enlivens and secures the running of a blockchain. In other word, consensus protocol plays an important role on making blockchain be a trusted decentralized public leger.

Particularly, we build the consensus protocol on DeepBC stemmed from Algorand [41], [42] which is a promising blockwise-BA protocol. There exist three main steps: (1) To randomly select a leader who creates a new block by calling cryptographic sortition. (2) A committee verifies and agrees on the new block by executing a Byzantine agreement protocol. The committee is constituted by transaction participants whose transactions are included inside the new block. (3) Each verifier in (2) step tells neighbors the new block via gossip protocol so that the new block is known by all participants on DeepBC. The consensus protocol on DeepBC demonstrates three properties including safety, correctness and liveness, which guarantees the health of DeepBC. Specifically, safety means that all honest parties agree on the same transaction history on DeepBC. Correctness represents that any transaction agreed by any honest party comes from a honest party. Last, liveness says that parties and workers are willing to continuously act on DeepBC, thereby keeping DeepBC living. In order to achieve these three properties, we assume that DeepBC enables synchronous message transmission. With the synchronous network assumption, all parties agree on a chain with the most asserts. With the implementation of step (1), we also assume at most available 2/3 \$$Coin$ are possessed by honest parties. We define that r_i is referred to the round which creates the block $block_i$.

- **Leader selection** A leader selection means that a block selection. At the round r_i, a leader $leader_i$ is randomly chosen from workers who create the block $block_i$. Before r_i beginning, which $leader_i$ being chosen is unpredictable and random, and after r_i ending, $leader_i$ is public. We call the sortition method of Algorand as form $Algorand.Sortition(sk, seed_i, \tau = 1, role = worker, w, W) \longrightarrow < hash, \pi, j >$ and $Algorand.VerifySort(pk, hash, \pi, seed_i, \tau, role = worker, w, W) \longrightarrow j$ for leader selection and leader verification, respectively. Specifically, a pair of sk and pk is owning to a participant. $seed_i$ is random selected based on $seed_{i-1}$ that is $seed_i = H(seed_{i-1}||r_i)$. $\tau = 1$ means we select only one leader from workers $role = worker$.

Algorithm 3: F^*_{ct} where ct means collaborative training.

Gradient Collecting

Wait to receive a message (input, $sid, T_t, pk^{psu}_{P_j}, C(\triangle\mathbf{W}), Proof_{PK_j}, d(\$Coin)$) from $pk^{psu}_{P_j}$ for all $j \in \{1, ..., N\}$. Assert time $T_t < T_{t_1}$. Here, sid means session identifier and $d(\$Coin)$ means amount of deposits. Then, wait to receive a message (input, $sid, T_t, pk^{psu}_{P_j} \in \mathbb{C}, C(\triangle\mathbf{W}), Proof_{PK_j}$, H', $h' \times d(\$Coin)$) from $\mathcal{S}$ (adversary). Assert time $T_t < T_{t_1}$. Here, H' means the set of the remaining honest parties and $|H'| = h'$.

Compute $fsver_1(C(\triangle\mathbf{W}), Proof_{PK_j})$ for all $pk^{psu}_{P_j}$ where $j \in \{1, ..., N\}$.

Record the correct parties as $\{1, ..., N\} \setminus \mathbb{C}'$ according to the computation results.

Send(return, $d(\$Coin)$) to $pk^{psu}_{P_j}$ for all $j \in \{1, ..., N\} \setminus \mathbb{C}'$ after T_{t_1};

If $\mathcal{S}$ returns (continue, H'') where H'' means $H' \setminus \mathbb{C}'$, then send (output, Yes or No) to all $pk^{psu}_{P_j}$ where $j \in \{1, ..., N\}$, and send (payback, $(h - h'')d(\$Coin)$) to S where $|H''| = h''$, and send (extrapay, $d(\$Coin)$) to $pk^{psu}_{P_j}$ where $j \in H''$.

Else if S returns (abort), send (penalty, $d(\$Coin)$) to $pk^{psu}_{P_j}$ for all $j \in \{1, ..., N\}$.

Collaborative decryption

Wait to receive a message (input, $sid, T_t, pk^{psu}_{P_j}, C, C_j, Proof_{CD_j}, d(\$Coin)$) from $pk^{psu}_{P_j}$ for all $j \in \{1, ..., N\}$. Assert time $T_t < T_{t_5}$. Then, wait to receive a message (input, $sid, T_t, pk^{psu}_{P_j} \in \mathbb{C}, C, C_j, Proof_{CD_j}$, H', $h' * d(\$Coin)$) from $\mathcal{S}$ (adversary). Assert time $T_t < T_{t_5}$.

Compute $fsver_2(C, C_j, Proof_{CD_j})$ for all $pk^{psu}_{P_j}$ where $j \in \{1, ..., N\}$.

Record the correct parties as $\{1, ..., N\} \setminus \mathbb{C}'$ according to the computation result.

Send(return, $d(\$Coin)$) to $pk^{psu}_{P_j}$ for all $j \in \{1, ..., N\} \setminus \mathbb{C}'$ after T_{t_5};

If $\mathcal{S}$ returns (continue, H'') where H'' means $H' \setminus \mathbb{C}'$, then send (output, Yes or No) to all $pk^{psu}_{P_j}$ where $j \in \{1, ..., N\}$, and send (payback, $(h - h'')d(\$Coin)$) to S where $|H''| = h''$, and send (extrapay, $d(\$Coin)$) to $pk^{psu}_{P_j}$ where $j \in H''$.

Else if S returns (abort), send (penalty, $d(\$Coin)$) to $pk^{psu}_{P_j}$ for all $j \in \{1, ..., N\}$.

w means amounts of $\$Coins$ with the available validity value the participant possess and those $\$Coins$ without validity value do not be considered, which are different from Alogrand's. W presents the total amounts of $\$Coins$ on DeepBC. Thus, we can randomly select a leader and all participants also enable to verify the selected leader $leader_i$. We set the property of validity value is to limit the trend that is wealth accumulation. It may happen that participants become more rich by accumulating more money due to the higher probability being a leader.

• **Committee agreement** After leader verification, the block $block_i$ built by $leader_i$ is sent to the committees whose transactions are processed inside $block_i$. The participants in a committee verify the procession done by $leader_i$, i.e., to verify whether the update operation is right or not. If a committee recognizes it right, a signature on the $block_i$ signed by the committee. If not, the $block_i$ is rejected. Then, the $block_i$ is valid on DeepBC only if more than 2/3 committees sign and agree on the $block_i$. $leader_i$ gains $\$Coins$ from block rewards and $block_i$' transaction coins. Otherwise, the $block_i$ is abandoned, and an empty block as the new $block_i$ replacing the old $block_i$ is built on DeepBC. Meanwhile, the reputation of the leader decrease one value. Finally, the committees agree on the new $block_i$.

• **Neighbor gossip** The $block_i$ has been agreed on by the committees. In this step, participants in these committees are responsible to tell their neighbors the $block_i$ via the popular gossip protocol. Eventually, all participants make consensus on DeepBC.

5 Security Analysis

In this section, we recall our security goals in DeepBC which are presented in section 3. We further give our security analyses threefold with respect to three security goals.

• **Confidentiality guarantees for training gradients.** Recall that this security goal refers to protecting trading gradients of participants and parameters of a collaborative model from disclosing. For this goal, DeepBC employs Threshold Paillier algorithm which has the additive homomorphic property. We assume there exists a trusted setup, and the secret key cannot leak without the collaboration of at least t participants. We also assume at leat t participants are honest. Without loss of generality, both individual gradients and model parameters W are encrypted with the Threshold **Paillier.Encrypt** algorithm as the form of $C(W) = g^W_{model}(k)^n_{model}$. Based on the following lemma (derived from **Theorem 1** in [Fouque'00]), we state the confidentiality of individual gradients and model parameters is guaranteed.

Lemma 1. With the Decisional Composite Residuosity Assumption (DCRA) [Paillier'99] and in the random oracle model (served as $\mathcal{S}$), Threshold Paillier algorithm is t-robust semantically secure against active non-adaptive adversaries $\mathcal{A}$ with polynomial time attack power, if

$$|\mathbf{Pr}[(w_0, w_1) \leftarrow \mathcal{A}(1^\lambda, F^t(\cdot)); b \leftarrow \{0,1\}; C \leftarrow \mathcal{S}(1^\lambda, w_b); \mathcal{A}(C, 1^\lambda, F^t(\cdot)) = b] - 1/2| \leq negl(1^\lambda)$$

that is negligible in λ which is the system security parameter. In the lemma above, $F^t(\cdot)$ is used to represented that $\mathcal{A}$ controls at most t corrupted parties and learns their information including public parameters, the secret shares of the corrupted parties, the public verification keys, all the decryption shares and the validity of those shares. In addition, t-robust means that a Threshold Paillier ciphertext can be correctly decrypted, even if there exists $\mathcal{A}$ actively corrupts up to t parties. Semantic security is a general security proof methodology which depicts the security of an encryption algorithm, and in this setting, it depicts the confidentiality of encrypted information by the Threshold **Paillier.Encrypt** algorithm.

• **Public auditability for gradient collecting and parameter updating.** The security goal is that any party can audit the correctness of encrypted gradients and decryption shares during the processes of gradient collecting and parameter updating, respectively. Recall that we introduce the non-interactive zero-knowledge proof for these two processes in the setting of the Threshold Paillier algorithm, such as $fsprove_1$, $fsver_1$, $fsprove_2$, $fsver_2$, the methodology of which can be referred to the universally verifiable CDN (UVCDN) protocol [47]. Under the defined framework of UVCDN protocol, public auditability can be guaranteed if there exists a simulator that simulates the correctness proofs of honest parties and extracts witnesses of corrupted parties. We following demonstrate this statement with respect to the correctness proof of encrypted gradients by using ***Lemma*** 2 and ***Lemma*** 3. Similarly, the correctness proof of decryption shares can be discussed under the UVCDN framework, and we omit this part due to the space limitation.

Lemma 2. Given $X = C(x)$, $x = \triangle \mathbf{W}$, and $c \in \mathbb{C}$ where $\mathbb{C}$ is a finite set named the challenge space, Then,

$\{d \in_R Z_{n_{model}}; e \in_R Z^*_{n_{model}}; a := g^d_{model} e^{n_{model}} X^{-c}: (a; c; d, e)\}$

$\approx$

$\{a_1 \in_R Z_{n_{model}}; b_1 \in_R Z^*_{n_{model}}; a := g^{a_1}_{model} b_1^{n_{model}}; t := (a_1 + cx)/n_{model}; d := a_1 + cx; e := b_1 k^c_j g^t_{model}: (a; c; d, e)\}$

where $\approx$ denotes that the distributions are statistical indistinguishable.

Lemma 3. We define $X = C(x) = g^x_{model} r^{n_{model}}$, in which $x = \triangle \mathbf{W}, r = k$. Given $(a; s)$ which is generated by the announcement $\Sigma_{PK}.ann$, and c a challenge in respect to the announcement, there exists an extractor $\mathcal{E}$ can extract the witness of an adversary $\mathcal{A}$, if $\mathcal{A}$ can present two conversations for $(a; s)$, that is,

$|1 - \mathbf{Pr}[\mathcal{A}(X; x, r; a; s; c) \rightarrow (d, e; d', e'); \mathcal{E}(X; a; d, e; d', e') \rightarrow (x', r') = (x, r)]| \leq negl(1^\lambda)$

• **Fairness enhancement for collaborative training.** Recall that we employ two security mechanisms in the setting of Blockchain to enhance fairness for collaborative training. The two security mechanisms are trusted time clock and secure monetary penalty mechanisms. Based on the exact timestamp attached to each block which is decentralizedly maintained, to assume the trusted time clock mechanism makes sense. With the mechanism, behaviors in a contract are pushed to be accomplish ahead of a defined time point, which is demonstrated by the function $checkTimeout$ in the setting of DeepBC. On the other hand, we define two secure monetary penalty mechanisms we need, from which one is for gradient collecting and another one is for collaborative decryption. To explain these two mechanisms, we introduce a notion, secure computation with coins (SCC security) in a multi-party N setting, which is defined and proven by [45], [46] in a hybrid model as following.

Lemma 4. Defined input z, security parameter λ, a distinguisher Z, ideal process $\texttt{IDEAL}$, ideal adversary S in IDEAL, and ideal function f; and meanwhile defined a protocol π which interact with ideal function g in a model with adversary A, Then,

$\{\texttt{IDEAL}_{f,S,Z}(\lambda, z)\}_\lambda \in \mathbb{N}, z \in 0, 1^*$

$\equiv_c$

$\{\texttt{HYBRID}_{g,\pi,A,Z}(\lambda, z)\}_\lambda \in \mathbb{N}, z \in 0, 1^*$

where $\equiv_c$ denotes that the distributions are computationally indistinguishable.

Lemma 5. Let π is a protocol and f s a multiparty function. We say that π securely computes f with penalties if π SCC-realizes the functionality f^*.

According to ***Lemma*** 5, we require a protocol π SSC-realizes F as F^* that means F^* achieves secure gradient collecting or collaborative decryption with penalties. With F^* and the trusted time clock mechanism, we intent to implement fairness for gradient collecting and collaborative decryption by F^*_{ct} mentioned by **Algorithm 4**.

6 Implementation and Evaluation

In this section, we present a implementation prototype for DeepChain, which demonstrates our feasibility. We first build the blockchain setting to simulate DeepChain. With this setting, nodes which are regarded as parties participate in trading, and interact with two defined crucial smart contracts (i.e., *Trading Contract* and *Processing Contract*), in which generated transactions are serialized on the blockchain.

First, we choose Corda V3.0 to simulate DeepChain for adaption and simplification. Corda project is created by R3CEV, as well as widely applied in bank, financial institutes and trading areas. It is a decentralized ledger which absorbs the features of Bitcoin and Ethereum while creating its characteristics, such as data sharing based on need-to-know basis, deconflicting transactions with pluggable notaries. A Corda network contains multiple notaries where the consensus protocol introduced in section 4.2.5 can be executed for them. Though we do not implement this in this paper, we make it for our further work. Without the loss of generation, we build nodes and classify them into two kinds, parties and workers. They constitute into the nodes of two CorDapps agreeing on the blockchain, in which we define different business logic in five components, such as Flows, States, Contracts, Services and Serialisation whitelists.

Second, we build the deep learning environment with the libraries: Python in version 3.6.4, numpy in version 1.14.0, and tensorflow in version 1.7.0. We select the popular MINIST dataset which has 55000 training data, 5000 verification data and 10000 testing data. Then, we split this dataset into multiple groups according to the number of parties. Our training model derives from CNN, the structure of which is: `Input` $\rightarrow$ `Conv` $\rightarrow$ `Maxpool` $\rightarrow$ `Fully Connected` $\rightarrow$ `Output`. The weights and bias parameter in Conv, Fully Connected and Output layers are $w_1 = (10, 1, 3, 3)$ and $b_1 = (10, 1)$, $w_2 = (1960, 128)$ and $b_2 = (1, 128)$, $w_3 = (128, 10)$ and $b_3 = (1, 10)$, respectively.

TABLE 2
Training configuration

Parameters	values
iteration	1500
epoch	1
learning rate	0.5
mini batch size	64

Additionally, other training parameters are configured as the table 2 shown.

Third, recalled that we employ Threshold Paillier encryption combining with the super increasing sequence. We set the number of bits of modulus n_{model} to 1024 bits. It is worth noting that before executing encryption algorithm, the weight matrixes are assembled as a vector, which makes only a cipher be generated corresponding to a party.

We implement the aforementioned building blocks with three modules, CordaDeepChain, TrainAlgorithm and CryptoSystem, respectively. We evaluated the feasibility of training on the simulated DeepChain in terms of encryption and training performance in a multi-party setting. First of all, we evaluate encryption performance with the implemented program on a desktop which is an Intel(R) Xeon(R) CPU machine with 3.30 GHz cores and 16 GB memory. Fig. 4 shows the size of cipher is a constant when we encrypt various amounts of gradients which means the number of elements in the vector to be encrypted. Then, Fig. 5 shows the throughput when the encrypt algorithm is executed.

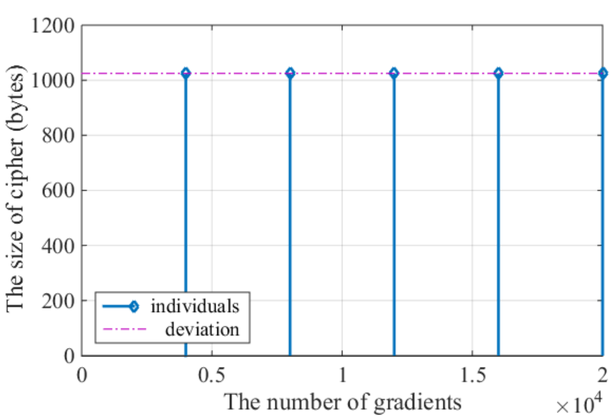

Fig. 4. Evaluation on the cipher size.

On the other hand, we create four parties participating in collaborative training and trading. Each party trains the local model with the training dataset which has the size of 13750 (by 55000/4). Then, single party gains the averaged gradients shared from the other three parties. we also create an external party only training on 13750-size dataset without the sharing averaged gradients, which is regarded as a baseline party. Through making the training accuracy comparison between the results from collaborative parties and the base line party, We demonstrate the accuracy improvement for single collaborative parties. The comparison result is shown in Fig 6.

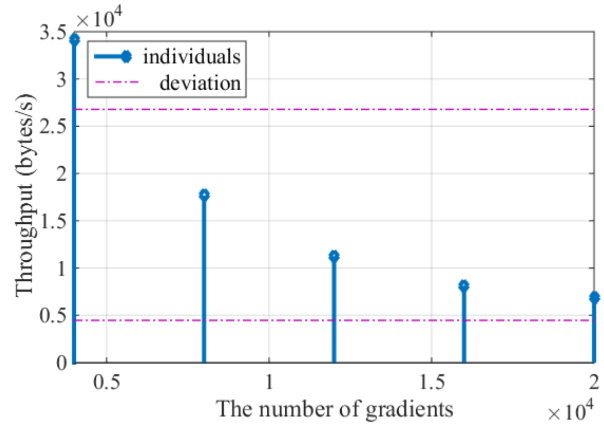

Fig. 5. Evaluation on the encryption throughput.

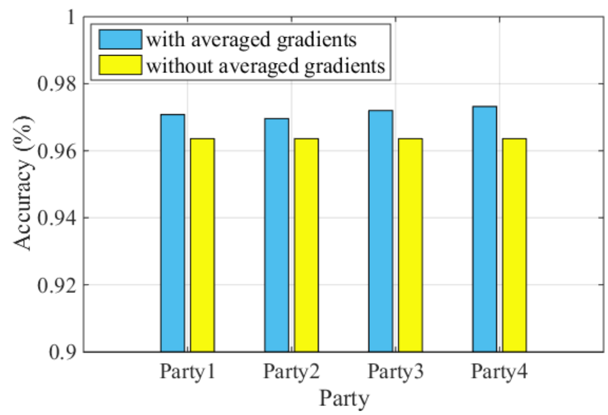

Fig. 6. The comparison on the training accuracy.

7 Conclusion and Future Work

In this paper, we present DeepChain, which is a healthy and win-win decentralized platform based on Blockchain for secure deep learning training. In the setting of federal learning, we introduce an incentive mechanism and meanwhile focus three security goals that are confidentiality, auditability as well as fairness. In addition, we claim the value of DeepChain in a long-term way. DeepChain stores training models where not only iterative training parameters but also trained models are recorded. On the one hand, it is obvious that trained models create financial values when the model-based pricing market is promising. This brings the owners of trained models with long-term benefits, since their models can serve for those who have AI tasks by the way of payment. On the other hand, all training processes and well-trained models are recorded, which could advance the development of transfer learning. Andrew Ng, in NIPS 2016 tutorial has said: "Transfer learning will be the next driver of ML success." [48] Thus, we take the first-step consideration that DeepChain can extend model values to transfer learning. Trained models which have gained knowledge solving one problem can be applied to a different but related problem. Then, the security problem, such as the privacy issue can be modeled, and in the case of model-to-model this issue could be discussed in the future work.

ACKNOWLEDGEMENT

This work was supported by National Key R&D Plan of China (Grant No. 2017YFB0802203), National Natural Science Foundation of China (Grant Nos. U1736203, 61732021, 61472165 and 61373158), Guangdong Provincial Engineering Technology Research Center on Network Security Detection and Defence (Grant No. 2014B090904067), Guangdong Provincial Special Funds for Applied Technology Research and Development and Transformation of Important Scientific and Technological Achieve (Grant No. 2016B010124009), the Zhuhai Top Discipline–Information Security, Guangzhou Key Laboratory of Data Security and Privacy Preserving, Guangdong Key Laboratory of Data Security and Privacy Preserving, National Joint Engineering Research Center of Network Security Detection and Protection Technology.

REFERENCES

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath et al., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," IEEE Signal processing magazine, vol. 29, no. 6, pp. 82–97, 2012.

[2] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?" IEEE Transactions on Image Processing, vol. 24, no. 12, pp. 5017–5032, 2015.

[3] E. Gawehn, J. A. Hiss, and G. Schneider, "Deep learning in drug discovery," Molecular informatics, vol. 35, no. 1, pp. 3–14, 2016.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," nature, vol. 521, no. 7553, p. 436, 2015.

[5] P. Danaee, R. Ghaeini, and D. A. Hendrix, "A deep learning approach for cancer detection and relevant gene identification," in PACIFIC SYMPOSIUM ON BIOCOMPUTING 2017. World Scientific, 2017, pp. 219–229.

[6] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in Data Mining (ICDM), 2016 IEEE 16th International Conference on. IEEE, 2016, pp. 171–180.

[7] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: building an efficient and scalable deep learning training system," in Usenix Conference on Operating Systems Design and Implementation, 2016, pp. 571–582.

[8] T. Chen and S. Zhong, "Privacy-preserving backpropagation neural network learning," IEEE Transactions on Neural Networks, vol. 20, no. 10, p. 1554, 2009.

[9] A. Bansal, T. Chen, and S. Zhong, "Privacy preserving back-propagation neural network learning over arbitrarily partitioned data," Neural Computing Applications, vol. 20, no. 1, pp. 143–150, 2011.

[10] J. Yuan and S. Yu, "Privacy preserving back-propagation learning made practical with cloud computing," IEEE Transactions on Parallel Distributed Systems, vol. 25, no. 1, pp. 212–221, 2014.

[11] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in Allerton Conference on Communication, Control, and Computing, 2015, pp. 909–910.

[12] P. Li, J. Li, Z. Huang, C. Z. Gao, W. B. Chen, and K. Chen, "Privacy-preserving outsourced classification in cloud computing," Cluster Computing, no. 1, pp. 1–10, 2017.

[13] Q. Zhang, L. Yang, and Z. Chen, "Privacy preserving deep computation model on cloud for big data feature learning," IEEE Transactions on Computers, vol. 65, no. 5, pp. 1351–1362, 2016.

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

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

[16] Y. Aono, T. Hayashi, L. Wang, S. Moriai et al., "Privacy-preserving deep learning via additively homomorphic encryption," IEEE Transactions on Information Forensics and Security, vol. 13, no. 5, pp. 1333–1345, 2018.

[17] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017, pp. 587–601.

[18] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Inference attacks against collaborative learning," arXiv preprint arXiv:1805.04049, 2018.

[19] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the gan: information leakage from collaborative deep learning," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017, pp. 603–618.

[20] T. Orekondy, S. J. Oh, B. Schiele, and M. Fritz, "Understanding and controlling user linkability in decentralized learning," arXiv preprint arXiv:1805.05838, 2018.

[21] A. Pyrgelis, C. Troncoso, and E. De Cristofaro, "Knock knock, who's there? membership inference on aggregate location data," arXiv preprint arXiv:1708.06145, 2017.

[22] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," arXiv preprint arXiv:1807.00459, 2018.

[23] "Health insurance portability and accountability act," https://www.hhs.gov/hipaa/index.html.

[24] G. Heigold, V. Vanhoucke, A. Senior, P. Nguyen, M. Ranzato, M. Devin, and J. Dean, "Multilingual acoustic models using distributed deep neural networks," in Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013, pp. 8619–8623.

[25] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[26] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014, pp. 459–474.

[27] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, 2013, pp. 397–411.

[28] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in Security and Privacy (SP), 2016 IEEE Symposium on. IEEE, 2016, pp. 839–858.

[29] S. Haykin, Neural networks: a comprehensive foundation. Prentice Hall PTR, 1994.

[30] J. Dean, G. Corrado, Monga et al., "Large scale distributed deep networks," in Advances in neural information processing systems, 2012, pp. 1223–1231.

[31] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," arXiv preprint arXiv:1412.7580, 2014.

[32] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," arXiv preprint arXiv:1501.02876, vol. 7, no. 8, 2015.

[33] M. Lin, S. Li, X. Luo, and S. Yan, "Purine: A bi-graph based deep learning framework," arXiv preprint arXiv:1412.6249, 2014.

[34] H. Cui, G. R. Ganger, and P. B. Gibbons, "Scalable deep learning on distributed gpus with a gpu-specialized parameter server," pp. 1–16, 2016.

[35] H. Ma, F. Mao, and G. W. Taylor, "Theano-mpi: A theano-based distributed training framework," CoRR, pp. 800–813, 2016.

[36] Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters.

[37] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, and S. Pollin, "Distributed deep learning models for wireless signal classification with low-cost spectrum sensors," CoRR, vol. abs/1707.08908, 2017.

[38] Distributed deep learning on edge-devices: Feasibility via adaptive compression, 2017.

[39] L. Chen, P. Koutris, and A. Kumar, "Model-based pricing for machine learning in a data marketplace," arXiv preprint arXiv:1805.11450, 2018.

[40] A. B. Kurtulmus and K. Daniel, "Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain," arXiv preprint arXiv:1802.10185, 2018.

[41] S. Micali, "Algorand: The efficient and democratic ledger," arXiv preprint arXiv:1607.01341, 2016.
[42] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017, pp. 51–68.
[43] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya, "Incentivizing outsourced computation," in Proceedings of the 3rd international workshop on Economics of networked systems. ACM, 2008, pp. 85–90.
[44] T. Nishide and K. Sakurai, "Distributed paillier cryptosystem without trusted dealer," in International Workshop on Information Security Applications. Springer, 2010, pp. 44–60.
[45] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in International Cryptology Conference. Springer, 2014, pp. 421–439.
[46] R. Kumaresan and I. Bentov, "How to use bitcoin to incentivize correct computations," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2014, pp. 30–41.
[47] B. Schoenmakers and M. Veeningen, "Universally verifiable multiparty computation from threshold homomorphic cryptosystems," in International Conference on Applied Cryptography and Network Security. Springer, 2015, pp. 3–22.
[48] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," arXiv preprint arXiv:1701.00160, 2016.