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

Jiasi Weng, Jian Weng, *Member, IEEE*, Jilian Zhang, Ming Li, Yue Zhang, and Weiqi Luo

Abstract—Deep learning can achieve higher accuracy than traditional machine learning algorithms in a variety of machine learning tasks. Recently, privacy-preserving deep learning has drawn tremendous attention from information security community, in which neither training data nor the training model is expected to be exposed. Federated learning is a popular learning mechanism, where multiple parties upload local gradients to a server and the server updates model parameters with the collected gradients. However, there are many security problems neglected in federated learning, for example, the participants may behave incorrectly in gradient collecting or parameter updating, and the server may be malicious as well. In this article, we present a distributed, secure, and fair deep learning framework named DeepChain to solve these problems. DeepChain provides a value-driven incentive mechanism based on Blockchain to force the participants to behave correctly. Meanwhile, DeepChain guarantees data privacy for each participant and provides auditability for the whole training process. We implement a prototype of DeepChain and conduct experiments on a real dataset for different settings, and the results show that our DeepChain is promising.

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* and *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 error is smaller than the pre-specified threshold.

This federated learning framework, however, cannot protect the privacy of the training data, even the training data is divided and stored separately. Some researchers show that the intermediate gradients can be used to infer important information about the training data [17], [18], for example. Shokri et al. [11] applied differential privacy technique by adding noises in the gradients to upload, achieving a tradeoff 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. [21] 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 confidentiality 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 authors are with the College of Information Science and Technology, Jinan University, and Guangdong/Guangzhou Key Laboratory of Data Security and Privacy Preserving, and National-Local Joint Engineering Research Center of Cyber Security Detection and Protection Technology, Guangzhou 510632, China. E-mail: {wengjiasi, cryptjweng, limjnu, zyueinfosec}@gmail.com, jilian.z.2007@smu.edu.sg, lwq@jnu.edu.cn.*

Manuscript received 4 Oct. 2018; revised 31 Mar. 2019; accepted 11 Oct. 2019. Date of publication 8 Nov. 2019; date of current version 1 Sept. 2021. (Corresponding author: Jian Weng.) Digital Object Identifier no. 10.1109/TDSC.2019.2952332

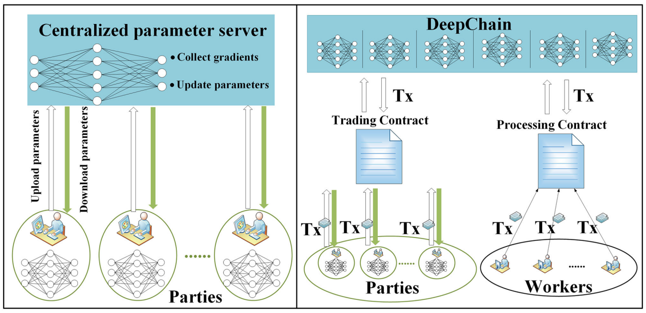

Fig. 1. The left corresponds to traditional distributed deep training framework, while the right is our DeepChain. Here, Trading Contract and Processing Contract are smart contract in DeepChain, together guiding the secure training process, while Tx refers to transaction.

the participants' local data. The second assumption, unfortunately, is no longer valid since membership inference attack on aggregated location data is now available [22].

Despite the fact that 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 updating 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. [23] 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 the confidentiality of gradients, but also the auditability of the correctness of gradient collecting and parameter updating.

The second problem is that in existing schemes the 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 [24], people's unwillingness to share and malicious attacks like identifying inference attacks against HCUPnet [25]. As a consequence, lack of training data will result in poor deep learning models in general [26]. 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]. Thus, 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.

Traditional incentive mechanisms include reputation-based [27], tit-for-tat [28] and payment-based mechanism [29], [30]. Usually, these mechanisms, except for tit-for-tat, need a trusted centralized authority to audit participant behaviors and arbitrate their payoff. Unfortunately, they fail to provide public auditability and decision fairness [28]. Although there is no trusted centralized party in tit-for-tat, it is not suitable for our setting, because a party's contribution is not symmetric to that of her counterparts. It is worth noting that Blockchain, originated from decentralized currency-system, enables distrustful nodes to share a common transaction ledger without the need of a trusted third party, by employing a consensus protocol and financial incentives. This motivates us to introduce a payment-based incentive mechanism that guarantees public authority and fairness.

In this paper, we propose DeepChain, a secure and decentralized framework based on Blockchain 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 updating are performed by workers (an entity in DeepChain that will be defined shortly) who are incented to process the transactions. To summarize, in this paper we made the following contributions:

- 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 updating, thus maintaining fairness during collaboration training.
- We implement DeepChain prototype and evaluate its performance in terms of cipher size, throughput, training accuracy and training time. We believe that DeepChain can benefit AI and machine learning communities, for example, it can audit the collaborative training process and the trained model, which represents the learned knowledge. Well-trained models can be used for paid services when the model-based pricing market is mature. In addition, 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 has arisen a surge of interests both in research community and industry [31]. It is an emerging technology as a decentralized, immutable, sharing and time-ordered ledger. Transactions are stored in blocks that contain timestamps and references (i.e., the hash of previous block), which are maintained as a chain of blocks. In Bitcoin, transactions which imply money transfers are created by pseudonymous participants and competitively collected to build a new block by an entity called *worker*. The worker who generates a new and valid block can gain some amount of rewards, hence the chain is continuously lengthened by workers. To achieve this, proof of work (PoW)-based consensus protocol and incentive mechanism are required.

There are a wide variety of consensus protocols, such as proof of stake (PoS)-based, byzantine fault tolerance (BFT)-based and hybrid protocols. In general, when introducing a new consensus protocol for a Blockchain setting, one needs to consider six problems: (1) *leader selection*, i.e., how to select a new block leader in each round; (2) *network model*, i.e., the message communication mode, such as asynchronous, synchronous and semi-synchronous; (3) *system model*, i.e., permissioned or permissionless system model, explaining whether a party can join the system freely; (4) *communication complexity*, reflecting the communication cost to propagate a new block to all parties in the system in each round; (5) *adversary assumption*, defining the probability of tolerating fault parties in the system; (6) *consensus property*, corresponding to the Agreement-Validity-Termination properties defined in classic consensus protocols [32].

The latest Algorand protocol [33], [34] is a hybrid consensus protocol based on PoS and BFT. Different from PoW-based consensus protocol, Algorand can guarantee consensus finality with overwhelming probability in terms of consensus property. Here, consensus finality means that a valid block appended to the chain will never be removed in the future, which is especially suitable for our problem. Without block data abandonment, we avoid spending excessive time and computation power to retrain a huge model. Also, Algorand protocol works in permissioned environment with the assumption of a synchronous network, which can be adapted to our setting.

Some latest Blockchain techniques, such as Ethereum and Hyperledger, introduce smart contract that supports Turing-complete programmability. These techniques are used to solve specific security issues in different application scenarios such as software-update management [35], cloud storage [36], [37] and machine learning [38]. On the other hand, a series of work on transaction privacy apply cryptographic tools in Blockchain, such as Zerocash [39], Zerocoin [40] and Hawk [41]. In general, consensus protocol and incentive mechanism in Blockchain are key ingredients for us to solve our problems, i.e., absence of incentive function and collaboration fairness guarantee.

2.2 Deep Learning and Distributed Deep Learning

2.2.1 Deep Learning

Deep learning enables a deep learning model to learn the abstractive representations of data. 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 number of neurons, and neurons at different layers can learn hierarchical features of the input training data, which represents 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 a weight assigned 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. These weights and bias are called *model parameters*, which need to be learned during the training.

Back-Propagation (BP) [42] 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 neural network training is called *activation*, which is the output of each neuron. Activation is used to learn non-linear features of inputs via function $Act(\cdot)$. To compute the output value of a neuron i at layer l, $Act(\cdot)$ takes all the n inputs of i from layer $l-1$ as the input. In addition, we assume that weight $w_{j,i}$ is associated with the connection between neurons j at layer $l-1$ and neurons i at layer l, and b_i is the bias of neuron i. Then, the value of neuron i at layer l can be obtained by $Act_i(l) = Act_i(\Sigma_{j=1}^{n}(w_{j,i} * Act_j(l-1)) + b_i)$.

The back-propagation step employs gradient descent method, which gradually reduces the model error E_{total}, i.e., the gap between model output value V_{output} and the target value V_{target}. Assume that there are n output units at the output layer. Then, the gap can be calculated by $E_{total} = \frac{1}{2}\Sigma_{i=1}^{n}(V_{target_i} - V_{output_i})^2$. Once E_{total} is available, weights $w_{j,i}$ can be updated through $w_{j,i} = w_{j,i} - \eta * \frac{\partial E_{total}}{\partial w_{j,i}}$, where η is the learning rate and $\frac{\partial E_{total}}{\partial w_{j,i}}$ is the partial derivative of E_{total} with respect to $w_{j,i}$. This is the main idea of gradient descent method. The learning process repeats until the pre-specified number of iterations to train is reached.

2.2.2 Distributed Deep Learning

When training a complex and multi-layer deep learning model, the aforementioned training procedure requires high computational overhead. To alleviate this problem, distributed deep learning training has been proposed recently, and some research work [43], [44], [45], [46], [47] and system implementations have been around, such as DistBelief [48], Torch [49], DeepImage [50] and Purine [51]. Generally, there are two approaches for distributed training, namely, model parallelism and data parallelism, where the former partitions a training model among multiple machines and the latter splits up the whole training dataset.

Our work focuses on the data parallelism approach, i.e., we have multiple machines and each machine maintains a copy of the training model while keeping a subset of the whole dataset as model input. These machines share the same parameters of the training model, by uploading/downloading parameters to/from a centralized parameter server. Then, machines upload their local training gradients, based on which the training model is updated by using SGD

(Stochastic Gradient Descent). They download updated parameters from the parameter server and continue to train the local model. This process repeats until machines obtain the final trained model.

3 Threats and Security Goals

In this section, we discuss threats to collaborative learning, and security goals that DeepChain 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, adversaries still 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. Here, collaborative decryption means that 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, which makes the honest ones suffer losses. All these malicious behaviors may fail the collaborative training task.

Security Goal 1: Auditability of Gradient Collecting and Parameter Updating. In DeepChain, assume that majority of the participants and at least $\frac{2}{3}$ of the workers are honest in gradient collecting and parameter updating, 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 updating, 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 at least $\frac{2}{3}$ 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, any third party can audit whether the decryption shares are correct or not.

Security Goal 2: Fairness Guarantee for 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 who behave 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. DeepChain achieves collaborative training by introducing incentive mechanism and transaction processing. In the process of collaborative training, data confidentiality and auditability are guaranteed by using cryptographic techniques.

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 gets her local gradients, she sends out the gradients by launching a transaction to 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 the 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 transaction 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 trade their gradients, and workers download and process the gradients. The processed gradients are then sent out by workers to the smart contract called *processing contract*. These correctly processed gradients are used to update parameters of the collaborative model by the leader selected from the workers. Parties download the updated parameters of the collaborative model and update their local models accordingly. After that parties begin the next iteration of model training.

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 that can be consumed in the future, for example, she may use rewards to pay for usage fee of trained models in DeepChain.

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 in 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 [38], [52]. 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 Section 4.2.2.

DeepChain combines together Blockchain techniques and cryptographic primitives to achieve secure, distributed and privacy-preserving deep learning. Suppose there are N parties P_j, $j \in \{1, \ldots, N\}$, and they agree on some predefined information such as a concrete collaborative model and initial parameters of the collaborative model. The information is attached to a transaction Tx_{co}^0 signed by all parties. Assume that the address corresponding to transaction Tx_{co}^0 is pk_{it_0}, where it_0 is the initial iteration. At the end of iteration i, the updated model in Tx_{co}^i 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 $Tx_{P_j}^i$, and all those transactions are collected by a *trading contract* at round 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 $\{Tx_{P_j}^i\}$ at round i have been collected, trading contract uploads them to DeepChain. After that, workers download those transactions $\{Tx_{P_j}^i\}$ 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 encrypted weight at round i in Tx_{co}^i, and $C(\mathbf{W}_{i+1})$ is the updated encrypted weights that will be attached to Tx_{co}^{i+1} for updating the local models in the next round $i+1$.

4.2 Building Blocks of DeepChain

DeepChain consists of two main building blocks including incentive mechanism and consensus protocol, which support the collaborative learning in DeepChain. The main procedures of collaborative learning will be introduced in the next subsection.

4.2.1 Incentive Mechanism

An incentive acts as a driving force for participants to actively and honestly take part in a collaborative training task, and the goal of incentive mechanism is to produce and distribute value, so that a participant gets rewards or penalties based on her contribution. The introduction of incentive mechanism is crucial for collaborative deep learning, due to the following reasons. First, for those parties who want a deep learning model but have insufficient data to train the model on their own, incentive can motivate them to join the collaborative training with their local data. Second, with reward and penalty, incentive mechanism ensures that (1) parties behave honestly in local model training and gradient trading, and (2) workers behave honestly in processing parties' transactions.

For ease of understanding the incentive mechanism, we give an example consisting of two parties. These two parties contribute their data to collaborative training by launching transactions. Suppose the data possessed by the two parties is not equal in quantity. Each party can launch transactions and pay transaction fee based on the amount of data she owned. Generally, the larger amount of data a party has, the less fee she will pay. The two parties agree on the total amount of fees for collaboratively training the model. The worker who successfully creates a new block when processing transactions can be the leader and earn the rewards. Note that transaction issuing and processing are verifiable, meaning that if some party poses an invalid transaction, the party would be punished, i.e., being fined. On the other hand, if a leader incorrectly processes a transaction, she will be punished accordingly. When collaborative training finished, parties themselves can benefit from the trained model that can bring revenue for them through charged services to those users who want to use the trained model.

To give a formal description of the incentive mechanism, we first introduce two properties, i.e., *compatibility* and *liveness* of the incentive mechanism for participants. Then, we further explain that parties and workers have incentive to behave honestly. Assume that we guarantee data privacy and security of the consensus protocol (explained in Section 4.2.2). We use v_c and v_i to denote the value of the collaboratively trained model and the trained individual model i, respectively, and we assume that v_c is greater than v_i.

First, we say the incentive mechanism exhibits compatibility if each participant can obtain the best result according to their contributions. Meanwhile, it has liveness only if each party is willing to update her local training model with value v_i by continuously launching transactions and each worker also has incentive to update the parameters of the collaborative training model with value v_c. Below we describe the importance of these two properties with respect to participant's true contribution and the corresponding payoff. As shown in Fig. 2, Let ω_P and ω_W be the contributions of a party and a worker to the final trained model, respectively, and π_P and π_W be their corresponding payoffs, respectively. At first, we assume that participant's contribution originates

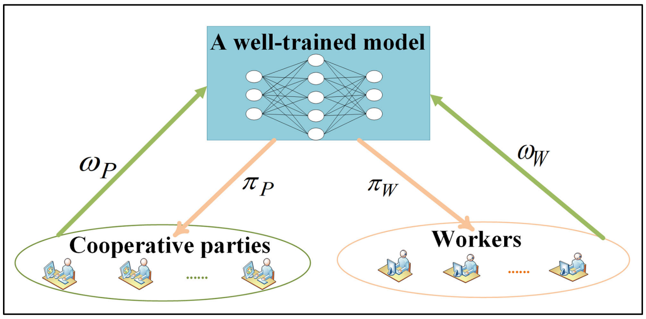

Fig. 2. The incentive mechanism of DeepChain, where ω_P and ω_W represent the contribution of a party and a worker for maintaining v_c, respectively, and π_P and π_W represent their payoffs, respectively.

from her correct behaviors with a high probability, and later we will explain that this assumption is reasonable.

Liveness. both the party and the worker have the same common interest to obtain a trained collaborative model. Because if a party costs v_i during the whole training process, then she would gain v_c in the end, which is attractive for her because v_c is greater than v_i. On the other hand, a worker will process transactions for collaboratively constructing the training model in order to earn rewards with probability, with which she could pay for the deep learning services in DeepChain. Note that the probability a worker obtains reward depends on the quantity of rewards she has already earned. The larger the quantity, the higher the probability she can get reward. As a result, both the party and the worker are incented to build the collaborative training model.

Compatibility. the more a party contributes ω_P, the more she will gain π_P. This holds for a worker too. During the collaborative training process, both party and worker are incentivized to do their best to contribute to building a training model $Max(\omega_P) \bigwedge Max(\omega_W)$, where the maximum total payoff is $Max(\pi_P) + Max(\pi_W)$. If any participant did not perform well, i.e., $(\omega_P = 0) \bigvee (\omega_W = 0)$, then there is no reward, i.e., $(\pi_P = 0) \bigwedge (\pi_W = 0)$. Here, $\bigwedge$ means '*and*' and $\bigvee$ means '*or*'. So we have

Payoff=

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

Next, we explain the assumption that participant's contribution originates from her correct behaviors with a high probability. We show that each party or worker is value-driven to behave correctly in each round so that she could obtain the highest payoff, if each of them is rational [53]. Yet, there may exist irrational malicious parties or workers who can behave incorrectly, but they would be punished if being caught. In our setting, we only consider fully-independent malicious behaviors. Other complex malicious behaviors can refer to the work [52]. If the probability that a party's behavior is correct is $Pr_c(P)$, then the corresponding value is $Value(Pr_c(P))$. Clearly, if the party's behavior is correct with probability $Pr_c(P)$=1, then she will obtain the highest value, i.e., $Value(1)$. Similarly, a worker can get value $Value(Pr_c(W))$ if she behaves correctly with probability $Pr_c(W)$. Assume that a method verifies a party's malicious behavior to be correct with probability $Pr_v(P)$, then the probability that a dishonest party is caught is $Pr_{vc}(P) = Pr_v(P) * (1 - Pr_c(P))$. Once the dishonest party is caught, she is punished by forfeiting her deposit and the loss is denoted as f_P, i.e, a fine.

Thus, the final value according to the party's correct behavior can be computed as

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

where $Pr_{vc}(P) = Pr_v(P) * (1 - Pr_c(P))$. The above value reaches maximum only when the party behaves honestly, i.e., $Pr_c(P) = 1$. Therefore, $Value(1) = \pi_P - \omega_P(1)$ holds. This indicates the importance of the incentive mechanism. Specifically, the values of $Pr_v(P)$, π_P, and f_P can be determined through the following theorems.

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

Proof. We need to prove that for any $Pr_c'(P) < \theta$, $Value(Pr_c'(P))$ is smaller than $Value(\theta)$. Without the loss of generality, we prove that for any $Pr_c'(P) < \theta$, we have $Value(Pr_c'(P)) < 0$. In other words, we have $Value\ (Pr_c'(P)) = \pi_P * (1 - Pr_{vc}'(P)) - f_P * Pr_{vc}'(P) - \omega_P(Pr_c'(P)) < 0$. When we set $f_P/\pi_P > 1/Pr_{vc}'(P) - 1$, then we have $\pi_P * (1 - Pr_{vc}'(P)) - f_P * Pr_{vc}'(P) < 0$. Thus, $Value(Pr_c'(P)) < 0$ holds.

Even if there are $\frac{1}{3}$ fully-colluding malicious parties (adapted to the security threshold of DeepChain), f_P/π_P still can be properly set under a practical value, i.e., no more than 2, which can refer to the implementation part in work [54]. Also, it is worth noting that blockchain here can serve as a trust bank guaranteeing the fair distribution of rewards and fines.

For a worker, analysis of the incentive mechanism is similar to the above analysis for a party, expect that the worker's payoff is obtained with probability. We denote this probability by Pr_{leader}, then we could determine the relationship between the four values Pr_{leader}, $Pr_v(W)$, π_W, and f_W by the following theorem, so as to encourage a worker to be honest. □

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

Proof. The proof is similar to the proof of *Theorem* 1, so we omit it. □

4.2.2 Consensus Protocol

Consensus protocol is essential in DeepChain, since it enables all participants to make a consensus upon some event in a decentralized environment. In this section, we introduce blockwise-BA protocol of DeepChain, based on the work of Algorand [33], [34]. The blockwise-BA protocol includes three main steps — (1) A leader who creates a new block is randomly selected by using cryptographic sortition, (2) A committee, consisting of participants whose transactions are included in the new block, verifies and agrees on the new block by executing a Byzantine agreement protocol [32], and

TABLE 1
Summary of Notations

Notation	Description
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	a generator of group G_1
Z_q^*	{1,2, ..., q-1}
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
$model_{co}$	collaborative deep learning model (collaborative model for short) to train
Σ_{PK}	correctness proof for a ciphertex that is indeed encrypted by a party's public key
Σ_{CD}	correctness proof for a decryption share

(3) Each verifier in the committee tells his neighbors the new block by using a gossip protocol [55], [56], so that the new block is known to all participants in DeepChain.

Our consensus protocol possesses three properties, i.e., *safety*, *correctness* and *liveness*. In particular, safety means that all honest parties agree on a same transaction history in DeepChain, whereas correctness requires that any transaction agreed by a honest party comes from a honest party. Liveness says that parties and workers are willing to continuously perform in DeepChain, hence keeping DeepChain alive. Based on these three properties, we assume that message transmission is synchronous and there are no more than $\frac{1}{3}$ malicious parties. In this setting, all parties agree on a chain with the largest amount of assets. We give details of the three steps of our consensus protocol below. Suppose block $block_i$ is created at round r_i.

Leader Selection. At round r_i, a leader $leader_i$ is randomly chosen from workers who collect transactions and put them into block $block_i$. To choose a leader, we invoke the sortition function of Algorand [33], which includes two functions *leader selection* and *leader verification*, as follows.

$$Sortition(sk, seed_i, \tau = 1, role = worker, \\ w, w_{total}) \rightarrow \langle hash, \pi, j \rangle \\ VerifySort(pk, hash, \pi, seed_i, \tau, role = worker, \\ w, w_{total}) \rightarrow j.$$

Here, sk and pk are owned by workers, and $seed_i$ is a random seed selected based on $seed_{i-1}$, i.e., $seed_i = H(seed_{i-1}||r_i)$, where H is a hash function. $\tau = 1$ means that only one leader is selected from workers $role = worker$. w represents the amount of $\$Coins$ that the participant possesses. Parameter w_{total} is the total amount of $\$Coins$ in DeepChain. It is worth mentioning that w is crucial, because it is used to control the probability that worker can gain reward according to the amount of rewards she has already earned (see Section 4.2.1).

Our definition of w is different from that of Alogrand, in that in our Leader Selection w only contains $\$Coins$ that have available validity value, while those without validity value are not considered. In this way, we can eliminate the phenomenon of wealth accumulation, in which a rich participant may become richer because she has a higher probability than her peers to be chosen as the leader. Through the two functions, we can randomly select a leader and all participants can also verify whether the selected leader $leader_i$ is legitimate.

Committee Agreement. After leader verification, the selected block $block_i$ is sent to the committee. Each participant in the committee verifies the transactions processed by $leader_i$, i.e., to verify whether weight update operations are correct or not. If the committee admits that $block_i$ is right based on a majority voting policy, then participants sign $block_i$ on behalf of the committee; otherwise, $block_i$ is rejected. Note that $block_i$ is valid only if more than $\frac{2}{3}$ of the committee members signed and agreed on it. If $block_i$ is valid, then $leader_i$ gains $\$Coins$ from block reward and transaction coins of $block_i$; otherwise, $block_i$ is discarded and a new empty block is created to replace $block_i$ in DeepChain. This process repeats until the committee agrees on $block_i$.

Neighbor Gossip. Suppose $block_i$ has been agreed on by the committee, then participants in the committee are responsible for telling their neighbors $block_i$, by using the popular gossip protocol [55], [56]. Therefore, after this step all participants arrive at a consensus in DeepChain.

4.3 Procedures of DeepChain

In order to achieve privacy-preserving collaborative learning, we introduce three procedures, namely, DeepChain bootstrapping, asset statement and cooperative training.

4.3.1 DeepChain Bootstrapping

DeepChain bootstrapping consists of two steps, i.e., DeepCoin distribution and genesis block generation. Assume that all parties and workers have registered (i.e., have a valid account) in DeepChain, where each one uses an address pk that corresponds to a DeepCoin unit for launching a transaction.

In the first step, DeepCoin distribution realizes DeepCoin allocation among parties and workers, and initially each party or worker is allocated with the same amount of DeepCoins. Then in the second step, a genesis block is generated at round 0, which contains initial transactions recording ownership statements for each DeepCoin.

After the genesis block is created, a random seed $seed_0$ is also publicly known, which is randomly chosen by registered users through a routine for distributed random number generation. When DeepChain keeps running, at round i, $seed_{i-1}$ is used for generating $seed_i$. It is worth mentioning that these random seeds are crucial for DeepChain, because they guarantee randomness when selecting a leader to create a new block at each round. The idea of introducing random seeds is motivated by Algorand's cryptographic sortition [33], [34], and details is given in Section 4.2.2.

4.3.2 Asset Statement

For ease of presentation, we list related cryptographic notations used in this section in Table 1. A party needs to state her asset, which enables her to find cooperators and accomplish her deep learning task. Asset statement does not reveal the content of asset, since it is simply some description of the asset, e.g., what kind of deep learning tasks the

TABLE 2
Example of Threshold Configuration for Maximum Number of Adversaries

Number of parties	Maximum number of adversaries	Threshold
$n = 4$	1	$t \in \{2, 3, 4\}$
$n = 5$	1	$t \in \{2, 3, 4, 5\}$
$n = 6$	1	$t \in \{2, 3, 4, 5, 6\}$
$n = 7$	2	$t \in \{3, 4, 5, 6, 7\}$
$n = 8$	2	$t \in \{3, 4, 5, 6, 7, 8\}$

asset can be used for. Specifically, party P states an asset by sending an asset transaction, which will be introduced later.

We recall the formation of a transaction. Note that a transaction is launched by a pseudo public key address pk_P^{psu} generated by P according to her wish in the following form.

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

Here, n is an integer. P selects a secret key $sk_P \in Z_q^*$ and generates n public keys $g_i^{sk_P} \in G_1, i \in [1, n]$. q and g are pre-specified parameters, and g_i equals to g^{r_i}, where r_i is a random element in Z_q^*. Suppose that party P_1 sends a transaction with her address $pk_{P_1}^{psu}$ to state her asset $data\_P_1$ as follows

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

In this transaction, the first part in the braces consists of $pk_{data\_P_1}$ and $\sigma_{j\_P_1}$, which is the statement proof that party P_1 indeed possesses asset $H_1(data\_P_1)$ without leaking the content of $data\_P_1$. In particular, $\sigma_{j\_P_1}$ contains l components, where $data\_P_1$ is divided into l blocks represented by $data_{j\_P_1}$, $j \in [1, l]$. The second part “Keywords” is the description of the asset $data\_P_1$. In our implementation, “Keywords” is in JSON” form that includes four fields, i.e., data size, data format, data topic and data description. With this transaction $Tran_{P_1}$, P_1 can fulfill her asset statement. We assume that the first stated asset is authentic, which is reasonable in Blockchain.

4.3.3 Collaborative Training

Based on stated assets, parties who have similar deep learning task can constitute a collaborative group, and the collaborative training process consists of the following four steps.

Collaborative Group Establishment. According to similar “Keywords”, parties can establish a collaborative group. It is worth noting that parties may get more detailed information about “Keywords” through off-line interactions and this is not the focus of our paper. Before forming a collaborative group, parties can audit cooperators’ asset to ensure authenticity of the asset ownership. The auditing process can be done by using the method in [57], and we omit the details for brevity.

Suppose there are N ($N > 3$) parties P_1, P_2,..., P_N that constitute a group with pseudonymity, i.e., pseudo public keys $pk_{P_1}^{psu}, pk_{P_2}^{psu}, \dots, pk_{P_N}^{psu}$ and their corresponding secret keys $sk_{P_1}, sk_{P_2}, \dots, sk_{P_N}$ are kept in private, respectively. Since different party launches transactions using her own pseudo public key $pk_{P_i}^{psu}$, transactions signed by the corresponding secret key sk_{P_i} can be verified to ensure that those transactions are from the same cooperative party P_i.

Collaborative Information Commitment. After the collaborative group is formed, parties agree on the information for securely training a deep learning model. In this step, we assume that a trusted component (e.g., a trusted hardware like Intel SGX [58]) only takes part in the setup phase in Threshold Paillier algorithm [59], and it is not involved in any other process. If there does not exist such a trusted component, we can accomplish the setup phase by using a distributed method such as the one in [60]. Parties agree on the following information.

(1) Number of cooperative parties, N.

(2) Index of the current round, r.

(3) Parameters of Threshold Paillier algorithm.

We have the following equation

$$PK_{model} = (n_{model}, g_{model}, \theta = as, V = (v, \{v_i\}_{i \in [1, \dots, N]})),$$

where modulus n_{model} is the product of two selected safe primes, and $g_{model} \in Z_{n_{model}^2}^*$, $a, s, \theta, v, v_i \in Z_{n_{model}}^*$. And $SK_{model} = s$ is randomly divided into N parts, where $s = f(s_1 + \cdots + s_N)$ and f is a function of secret sharing protocol (i.e., Shamir’s secret sharing protocol [61]). Each party owns a proportion of secure key s_i as well as v and $\{v_i\}$, $i \in [1, \dots, N]$ are public verification information, where v_i corresponds to s_i. A threshold $t \in \{\frac{N}{3} + 1, \dots, N\}$ is set as such that at least t parties together can decrypt a cipher. Specifically, we give the threshold configuration with respect to the number of adversaries in Table 2.

Note that training gradients to be encrypted are vectors with multiple elements, i.e., $\triangle \mathbf{W_{i,j}} = (w_{i,j}^1, \dots, w_{i,j}^l)$ where the length of $\triangle \mathbf{W_{i,j}}$ is l, i is the index of current training iteration, and $j \in \{1, \dots, N\}$. Due to the problem of cipher expansion, we encrypt a vector into one cipher instead of multiple ciphers with respect to multiple elements. Suppose that each value $w_{i,j}^1, \dots, w_{i,j}^l$ is no larger than integer d, $d > 0$. We choose a l-length super increasing sequence $\vec{\alpha} = (\alpha_1 = 1, \dots, \alpha_l)$ that simultaneously meets conditions (1) $\sum_{l=1}^{i-1} \alpha_l \cdot N \cdot d < \alpha_i, i = 2, \dots, l$, and (2) $\sum_{i=1}^{l} \alpha_i \cdot N \cdot d < n_{model}$. We then compute $(g_{model}^1, \dots, g_{model}^l) = (g_{model}^{\alpha_1}, \dots, g_{model}^{\alpha_l})$.

(4) A collaborative model $model_{co}$ to be trained.

For a collaborative model $model_{co}$, parties agree on the training neural network, the training algorithms, and configurations of the network such as number of network layers, number of neurons per layer, size of mini-batch and number of iterations. Beside those information, they also reach a consensus on initial weights $\mathbf{W}_0$ of $model_{co}$. Note that weights $\mathbf{W}_i$ would be updated to $\mathbf{W}_{i+1}$ after the ith iteration of training. They protect $\mathbf{W}_0$ by applying **Paillier. Encrypt** algorithm, i.e., $C(\mathbf{W}_0) = g_{model}^{\mathbf{W}_0} \cdot (k_0)^{n_{model}}$, where k_0 is randomly selected from $Z_{n_{model}}^*$. Note that we compute $g_{model}^{\mathbf{W}_0}$ with the help of the chosen super increasing sequence, i.e., $g_{model}^{\mathbf{W}_0} = g_{model}^{\alpha_1 \cdot w_0^1 + \dots + \alpha_l \cdot w_0^l}$, so that we generate a cipher for weight vector $\mathbf{W}_0 = (w_0^1, \dots, w_0^l)$.

(5) A commitment on $SK_{model} = s$, with respect to PK_{model}.

Commitment $commit_{SK_{model}}$ is obtained by combining parties’ commitments on their secret shares s_i. Recall that r is the index number of the current round. We have

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

here, $||$ denotes concatenation.

(6) The initial weights $\mathbf{W}_{0,j}$ of local model of party j.

Each party provides her local model's initial weights that are encrypted by **Paillier.Encrypt** algorithm, i.e., $C(\mathbf{W}_{0,j}) = g_{model}^{\mathbf{W}_{0,j}} \cdot (k_j)^{n_{model}}$, where $k_j \in Z^*_{n_{model}}$, $j \in \{1, \ldots, N\}$.

(7) Amount of deposits $d(\$Coin)$.

Each cooperative party is required to commit some amount of deposits for secure computation. During collaborative training, if a party misbehaves on purpose, her deposit $d(\$Coin)$ would be forfeited and compensated for other honest parties. Otherwise, those deposits would be refunded after the training process finished.

All the above collaborative information are recorded in a transaction $Tran_{co}$ that is uploaded to DeepChain. Specifically, $Tran_{co}$ is in the following form and is attached to a commonly coordinated address pk_{co}^{psu}.

$$Tran_{co} = pk_{co}^{psu} \rightarrow \Big\{N, r, PK_{model}, d, \vec{\alpha}, model_{co},\\ commit_{SK_{model}}, C(\mathbf{W}_{0,j}), d(\$Coin)\Big\}.$$

In addition, two roles called *trader* and *manager* are defined for parties in a collaborative group, which will be explained shortly. Next we introduce how collaborative training is securely accomplished through the remaining two steps, namely, *Gradient collecting via Trading Contract* and *Parameter updating via Processing Contract*.

First of all, parties iteratively trade their gradients through *Trading Contracts* that are executed by a *manager* selected from cooperative parties. The trading gradients are honestly encrypted by each trader and meanwhile the correct proofs of encryption are attached that indicate two security requirements, i.e., confidentiality and auditability. Herein, we say gradient transactions are generated. In terms of confidentiality, if a trader does not disclose her gradients, then no one can gain information about the gradients. In addition, traders (at most t parties) need to cooperatively decrypt the updated parameters. Similar to [41], we assume that the manager does not disclose what she knows. In terms of auditability, there exist proofs of correct encryption which can be auditable. When cooperatively decrypting, each trader presents her own decryption proof. Those proofs are generated non-interactively and publicly auditable by any party in DeepChain.

Through timeout-checking and monetary penalty mechanism, behaviors of the traders and the manager are forced to be authentic and fair. Even if the manager colludes with traders, the outcome of *Trading Contract* cannot be modified [41]. In addition to *Trading Contract*, *Processing Contract* is responsible for parameter updating. Workers process transactions by adding up gradients, and send computation results to *Processing Contract*. *Processing Contract* verifies correct computation results and updates model parameters for the group. Note that once smart contract is defined, it can be automatically executed in response to some trigger event. In this setting, 'computation results sent to *Processing Contract*' is the trigger event, and *Processing Contract* has a predefined function to verify those computation results by the rule of majority voting. These two contracts are iteratively invoked, so as to accomplish the whole training process. Details of the two steps are given below.

Gradient Collecting via Trading Contract. As shown in Algorithm 1, *Trading Contract* invokes six functions, i.e., line 1, 4, 7, 10, 13 and 16 of Algorithm 1, for training $model_{co}$. At the end of each of the functions, we declare a time point T_{t_i} to check time-out events, and these six time points satisfy $T_{t_i} < T_{t_{i+1}}$, $i = 1, 2, \ldots, 5$. We set up the time points according to Greenwich Mean Time. The time interval between T_{t_1} and T_{t_6} can be determined according to the time interval between two consecutive training iterations, e.g., for iteration i and $i+1$, we have $|T_{t_6} - T_{t_1}| \leq |T_{i+1} - T_i|$.

By the end of a time point T_{t_i}, function *checkTimeout* checks whether the parties finish the events or not by T_{t_i}. If some party is caught, the monetary penalty mechanism will be performed to forfeit deposit of the party, and the failed step is re-executed. During collaborative training, the six time points are updated accordingly with iterations, e.g., $T'_{t_1} = T_{t_1} + |T_{i+1} - T_i|$.

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

1 **receiveGradientTX()**
2 checkTimeout(T_{t1})
3 updateTime() // $T'_{t_1} = T_{t_1} + |T_{i+1} - T_i|$
4 **verifyGradientTX()**
5 checkTimeout(T_{t2})
6 updateTime() // $T'_{t2} = T_{t2} + |T_{i+1} - T_i|$
7 **uploadGradientTX()**#attaching to the address pk_{co}^{psu}
8 checkTimeout(T_{t3})
9 updateTime() // $T'_{t3} = T_{t3} + |T_{i+1} - T_i|$
10 **downloadUpdatedParam()**#from the address pk_{co}^{psu}
11 checkTimeout(T_{t4})
12 updateTime() // $T'_{t4} = T_{t4} + |T_{i+1} - T_i|$
13 **decryptUpdatedParam()**
14 checkTimeout(T_{t5})
15 updateTime() // $T'_{t5} = T_{t5} + |T_{i+1} - T_i|$
16 **return()**
17 checkTimeout(T_{t6})
18 updateTime() // $T'_{t6} = T_{t6} + |T_{i+1} - T_i|$

Algorithm 1 works as follows. As shown in line 1, at the ith iteration each party P_j, $j \in \{1, \ldots, N\}$ sends a gradient transaction $Tran^i_{P_j}$ to $receiveGradientTX()$. A publicly auditable proof $Proof_{PK_{i,j}}$ is also attached to the transaction to guarantee encryption correctness. We have

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

Then in line 4, function $verifyGradientTX()$ verifies correctness of the encrypted gradients via function $fsver_1(\Sigma_{PK}; C(\triangle \mathbf{W}_{i,j}); Proof_{PK_{i,j}}; pk^{psu}_{P_j})$. Specifically, it verifies whether $C(\triangle \mathbf{W}_{i,j})$ is indeed the encryption of $\triangle \mathbf{W}_{i,j}$ with random number k_j. Here, $pk^{psu}_{P_j}$ can be regarded as the identity information attached to the proof, avoiding *replay attack* by a malicious party. In line 7, function $uploadGradientTX()$ uploads the transactions that have been verified successfully. When model parameter updating finished, $downloadUpdated Param()$ retrieves the latest parameters, as can be seen in

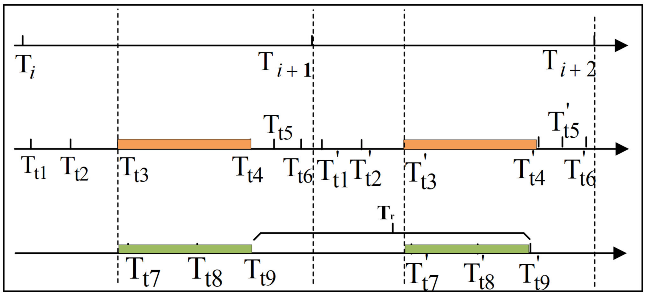

Fig. 3. Configuration of time points in Processing Contract. From top to bottom: (1) the timeline of collaborative training, (2) the timeline of trading (in Trading Contract), (3) the timeline of block creation. Here, the vertical orange bar refers to the interval between verified gradient transaction being uploaded and updated parameters to be downloaded. The vertical green bar refers to the interval between worker's transactions being sent and final updated result being uploaded.

line 10. Recall that *Processing Contract* computes gradients $\sum_{j=1}^{N} \triangle \mathbf{W}_{i,j}$ for model $model_{co}$.

Suppose that the latest iteration is i, the cipher of the latest parameters is $C(\mathbf{W}_i)$ and we denote it as C_i for brevity. Then $decryptUpdatedParam()$ collects parties' decryption shares on C_i for collaborative decryption, which generates $C_{i,j}$, $j \in 1, \ldots, N$. Meanwhile, the corresponding proofs for correct shares $Proof_{CD_{i,j}}$ are also provided, as follows.

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

The proof $Proof_{CD_{i,j}}$ is used to verify validity of the decryption shares, i.e., $\Delta s_j = log_{C_i^4}(C^2_{i,j}) = log_v(v_j)$, through function $fsver_2(\Sigma_{CD}; (C_i, C_{i,j}, v, v_j); Proof_{CD_{i,j}}; pk^{psu}_{P_j})$. If majority of the parties are honest, then C_i can be correctly recovered to plaintext by

$$((\Pi_{j\in H} C_{i,j}^{2\mu_j} - 1)/n_{model})(4\Delta^2\theta)^{-1} \bmod \quad n_{model},$$

where μ_j is the Lagrange interpolation coefficient with respect to P_j, and the plaintext is pushed to parties by function $return()$ in line 16.

Algorithm 2. Processing()

1 **updateTX()**
2 checkTimeout(T_{t7})
3 updateTime() // $T'_{t7} = T_{t7} + T_r$
4 **verifyTX()**
5 checkTimeout(T_{t8})
6 updateTime() // $T'_{t8} = T_{t8} + T_r$
7 **appendTX()**
8 checkTimeout(T_{t9})
9 updateTime() // $T'_{t9} = T_{t9} + T_r$

• *Parameter updating via Processing Contract.* Algorithm 2 summarizes the process of *Processing Contract*, which contains three functions, as shown in line 1, 4, and 7. Suppose that at the ith iteration of collaborative training, local gradients $C(\triangle \mathbf{W}_{i,j})$, $j \in \{1, \ldots, N\}$, have been uploaded, then workers competitively execute update operations by

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

Once update operation finished, workers then send the updated results through transactions to function $updateTX()$ in *Processing Contract*, as shown in line 1.

At the meantime, a leader is randomly chosen from the workers by using the consensus protocol of DeepChain. Note that at this moment we defer the reward to the leader until her computational work is verified by using function $verifyTX()$ as shown in line 4, that employs majority voting policy. In other words, the leader's computational result $C(\mathbf{W}_i)$ will be compared against those of the other workers, and her result is admitted only if the majority of the workers produce the same result. Otherwise, the leader would be punished according the monetary penalty mechanism and she gains no reward. In such case, we repeat the procedure to chosen a new leader from the remaining workers. The more often a worker is punished, the lower probability she will be chosen as a leader. Once we get a legitimate leader, her block with correctly updated result is appended to DeepChain through $appendTX()$, as shown in line 7.

In *Processing Contract*, time points T_{t7}, T_{t8} and T_{t9} will be updated to $T'_{t7} = T_{t7} + T_r, T'_{t8} = T_{t8} + T_r$ and $T'_{t9} = T_{t9} + T_r$, respectively, where T_r is the time needed to create a new block between consecutive rounds in DeepChain. Fig. 3 gives an example of time point configuration scheme to illustrate relationship of time points of the trading and processing contracts. Suppose that at the ith iteration, the time points are set as such that $T_{t_1} < T_{t_2} < T_{t_3} \le T_{t7} < T_{t8} < T_{t9} \le T_{t4} < T_{t_5} < T_{t_6}$. At the meantime, the relationship between the three time intervals is $T_r \le |T_{t_6} - T_{t_1}| \le |T_{i+1} - T_i|$.

Algorithm 3. $F^*_{GradientCollecting}$

1 Receive (*input*, $sid, T_t, pk^{psu}_{P_j}, C(\triangle\mathbf{W}), Proof_{PK_j}$, d(\$*Coin*)) from $pk^{psu}_{P_{j\in\{1,\ldots,N\}}}$. Assert time $T_t < T_{t_1}$. Receive (*input*, $sid, T_t, pk^{psu}_{P_{j\in\mathbb{C}}}$, $C(\triangle\mathbf{W}), Proof_{PK_j}$, H', $h' \times d(\$Coin)$) from $\mathcal{S}$. Assert time $T_t < T_{t_1}$.
2 Compute $fsver_1(C(\triangle\mathbf{W}), Proof_{PK_j})$ for $pk^{psu}_{P_{j\in\{1,\ldots,N\}}}$, and record $\{1, \ldots, N\} \setminus \mathbb{C}'$.
3 Send(*return*, $d(\$Coin)$) to $pk^{psu}_{P_{j\in\{1,\ldots,N\}\setminus\mathbb{C}'}}$ after T_{t_1}.
4 If $\mathcal{S}$ returns (continue, H''), then send (*output*, *Yes* or *No*) to $pk^{psu}_{P_{j\in\{1,\ldots,N\}}}$, and send (*payback*, $(h - h'')d(\$Coin)$) to $\mathcal{S}$, and send (*extrapay*, $d(\$Coin)$) to $pk^{psu}_{P_{j\in H''}}$, else if $\mathcal{S}$ returns (abort), send (*penalty*, $d(\$Coin)$) to $pk^{psu}_{P_{j\in\{1,\ldots,N\}}}$.

Algorithm 4. $F^*_{CollaborativeDecryption}$

1 Receive (*input*, $sid, T_t, pk^{psu}_{P_j}, C, C_j, Proof_{CD_j}, d(\$Coin)$) from $pk^{psu}_{P_{j\in\{1,\ldots,N\}}}$. Assert time $T_t < T_{t_5}$. Receive (input, sid, T_t, $pk^{psu}_{P_j} \in \mathbb{C}$, $C, C_j, Proof_{CD_j}$, H', $h' * d(\$Coin)$) from $\mathcal{S}$. Assert time $T_t < T_{t_5}$.
2 Compute $fsver_2(C, C_j, Proof_{CD_j})$ for $pk^{psu}_{P_{j\in\{1,\ldots,N\}}}$ and record $\{1, \ldots, N\} \setminus \mathbb{C}'$.
3 Send(*return*, $d(\$Coin)$) to $pk^{psu}_{P_{j\in\{1,\ldots,N\}\setminus\mathbb{C}'}}$ after T_{t_5};
4 If $\mathcal{S}$ returns (continue, H''), then send (*output*, *Yes* or *No*) to $pk^{psu}_{P_{j\in\{1,\ldots,N\}}}$, and send (*payback*, $(h - h'')d(\$Coin)$) to S, and send (*extrapay*, $d(\$Coin)$) to $pk^{psu}_{P_{j\in H''}}$, else if S returns (abort), send (*penalty*, $d(\$Coin)$) to $pk^{psu}_{P_{j\in\{1,\ldots,N\}}}$.

In addition to the above configuration scheme for time points, we employ secure monetary penalty mechanism to guarantee fairness in gradient collecting and collaborative decryption. Specifically, enlightened by the penalty mechanism proposed by Bentov et al. [62] and Kumaresan et al. [63], we design our secure monetary penalty mechanism based on *Trading Contract*, presented in Algorithms 3 and 4.

In particular, in *Gradient collecting* (Algorithm 3) fairness is guaranteed due to (1) honest collaborative parties must launch gradient transactions to be correctly verified before the pre-specified time point, and (2) dishonest parties who launch incorrect transactions or delayed transactions will be penalized, and the honest ones will be compensated for. In line 1, *Trading Contract* waits to receive a *input* message from $pk^{psu}_{P_j}$ for all $j = 1, \dots, N$ before time T_{t_1}. By defining $\mathbb{C} \subseteq \{1, \dots, N\}$ as adversarial parties $\mathcal{S}$ in the *input* step, the contract also waits an *input* message from $\mathcal{S}$. Here, sid is session identifier, $d(\$Coin)$ is deposit, and H' means the set of the remaining honest parties, where $|H'| = h'$. In line 2, the contract verifies the ciphertext for all $pk^{psu}_{P_j} \in H'$, and records the correct parties $\{1, \dots, N\} \setminus \mathbb{C}'$, where $\mathbb{C}'$ refers to corrupted parties in this step. In line 3, the contract sends *return* messages to $pk^{psu}_{P_j}$ for $j \in \{1, \dots, N\} \setminus \mathbb{C}'$. In line 4, we wait for a return message from $\mathcal{S}$. If the returned message is continue, then the contract outputs normally to all $pk^{psu}_{P_j}$ $(j \in \{1, \dots, N\})$, by sending *payback* message to $\mathcal{S}$ and *extrapay* to $pk^{psu}_{P_j}$ in H'', where $H'' = H' \setminus \mathbb{C}'$ and $|H''| = h''$; otherwise, the contract sends *penalty* to $pk^{psu}_{P_j}$, $j \in \{1, \dots, N\}$.

Similarly, fairness is also achieved in *Collaborative decryption* (Algorithm 4), since (1) a party who gives a correct decryption share no later than the pre-defined time point receives no penalty, and (2) if an adversary successfully decrypts the cipher but a legitimate party fails to do so, then the party should be compensated for.

5 Security Analysis

In this section, we revisit our security goals of DeepChain presented in section 3 and give security analysis accordingly.

5.1 Confidentiality Guarantee for Gradients

Confidentiality guarantees that models gradients do not be exposed. To achieve this goal, DeepChain employs Threshold Paillier algorithm that provide additive homomorphic property. We assume there exists a trusted setup (refer to Section 4.2.4) and the secret key cannot leak without collaboration of at least t participants. We also assume that at least t participants are honest. Without loss of generality, both local gradients and model parameters $\mathbf{W}$ are encrypted with the Threshold Paillier algorithm, $C(\mathbf{W}) = g^{\mathbf{W}}_{model}(k)^{n_{model}}$. Based on the following lemma that is derived from the work [59]'s Theorem 1, we can guarantee confidentiality of local gradients and model parameters.

Lemma 1. *With the Decisional Composite Residuosity Assumption (DCRA) [64] and the random oracle model $\mathcal{S}$, Threshold Paillier algorithm is t-robust semantically secure against active non-adaptive adversaries $\mathcal{A}$ with polynomial time power to attack, if the following are satisfied*

$$\begin{aligned}&\mathbf{Pr}\big[(w_0, w_1) \leftarrow \mathcal{A}(1^\lambda, F^t(\cdot));\\ &b \leftarrow \{0,1\};\\ &C \leftarrow \mathcal{S}(1^\lambda, w_b):\\ &\mathcal{A}(C, 1^\lambda, F^t(\cdot)) = b\big] \leq negl(1^\lambda) + \tfrac{1}{2}\end{aligned}$$

which indicates that the probability for adversaries to distinguish between w_0 and w_1 is negligible, in system security parameter λ. Here, $F^t(\cdot)$ means that $\mathcal{A}$ has at most t corrupted parties and $\mathcal{A}$ learns their information including public parameters, secret shares of the corrupted parties, public verification keys, all decryption shares and validity of those shares. In addition, t-robust means that a Threshold Paillier ciphertext can be correctly decrypted, even in the case that $\mathcal{A}$ can have up to t corrupted parties. Semantic security is a general security proof methodology to measure the security of an encryption algorithm and in our context it measures confidentiality of the encrypted information by using the Threshold Paillier algorithm.

5.2 Auditability of Gradient Collecting and Parameter Update

Auditability ensures that any third party can audit correctness of encrypted gradients and decryption shares in gradient collecting stage and parameter updating stage, receptively. We achieve auditability by following universally verifiable CDN (UVCDN) protocol [65]. Specifically, correctness proof provided in UVCDN protocol is based on $\sum$-protocols, where a possible malicious prover proves correctness to a honest verifier that he indeed knows a witness w for a certain statement v leading to $(v; w) \in R$, here R is a binary relation. Generally, a correctness proof consists of three protocols, namely, announcement (denoted as $\sum.ann$), response (denoted as $\sum.res$) and verification (denoted as $\sum.ver$), which are defined according to the purpose of correctness proof (i.e., R). We will give the correctness proof by using two procedures $fsprove$ and $fsver$ for $\sum.ann$, $\sum.res$ and $\sum.ver$.

Algorithm 5. $fsprove_1(\Sigma_{PK}; C(\triangle \mathbf{W}_{i,j}); \triangle \mathbf{W}_{i,j}, k_j; pk^{psu}_{P_j})$

#announcement
$\Sigma_{PK}.ann(C(\triangle \mathbf{W}_{i,j}); \triangle \mathbf{W}_{i,j}, k_j) :=$
$a_1 \in_R Z_{n_{model}}, b_1 \in_R Z^*_{n_{model}}, a = g^{a_1}_{model} b_1^{n_{model}};$
$(a; s) = (a; a_1, b_1)$
#challenge
$c = \mathcal{H}(C(\triangle \mathbf{W}_{i,j}) || a || pk^{psu}_{P_j})$
#response
$\Sigma_{PK}.res(C(\triangle \mathbf{W}_{i,j}); \triangle \mathbf{W}_{i,j}, k_j; a; s; c) :=$
$t = \frac{(a_1 + c \triangle \mathbf{W}_{i,j})}{n_{model}}, d = a_1 + c \triangle \mathbf{W}_{i,j},$
$e = b_1 k_j^c g^t_{model}; r = (d, e)$
return $Proof_{PK_{i,j}} := (a; c; r)$

Algorithm 6. $fsver_1(\Sigma_{PK}; C(\triangle \mathbf{W}_{i,j}); Proof_{PK_{i,j}}; pk^{psu}_{P_j})$

#verification
#$Proof_{PK_{i,j}} := (a; c; r), r = (d, e)$
$\Sigma_{PK}.ver(C(\triangle \mathbf{W}_{i,j}); a; c; r) :=$
$(c == \mathcal{H}(C(\triangle \mathbf{W}_{i,j}) || a || pk^{psu}_{P_j})) \bigwedge (g^d_{model} e^{n_{model}} == a(C(\triangle \mathbf{W}_{i,j}))^c)$
return Yes or No

In Section 4.2.4, to prove the correctness of encrypted gradients $R = \{(C(\triangle \mathbf{W}_{i,j}); \triangle \mathbf{W}_{i,j}, k_j)\}$ where $C(\triangle \mathbf{W}_{i,j}) = g_{model}^{\triangle \mathbf{W}_{i,j}} (k_j)^{n_{model}}$, a publicly auditable proof $Proof_{PK_{i,j}}$ is generated by procedure $fsprove_1$. Then, any party can execute procedure $fsver_1$, by taking $Proof_{PK_{i,j}}$ as input, to verify whether $C(\triangle \mathbf{W}_{i,j})$ is indeed the encryption of $\triangle \mathbf{W}_{i,j}$ with random number k_j under public key PK_j ($i \in \{1, \ldots, \#iteration\}$, $j \in \{1, \ldots, N\}$). The concrete procedures of $fsprove_1$ and $fsver_1$ for $\sum$-protocols are given in Algorithms 5 and 6, respectively. In addition, $\sum_{PK}$ refers to the $\sum$-protocols achieved by $fsprove_1$ and $fsver_1$. Correspondingly, $\sum_{CD}$ is for the $\sum$-protocols realized by $fsprove_2$ (Algorithm 7) and $fsver_2$ (Algorithm 8) for proving decryption correctness. Note that in Algorithms 6 and 8, '==' is used to test equality.

Algorithm 7. $fsprove_2(\Sigma_{CD}; C_i, C_{i,j}, v, v_j; \triangle s_j; pk_{P_j}^{psu})$

#announcement
1 $\Sigma_{CD}.ann(C_i, C_{i,j}, v, v_j; \triangle s_j) :=$
$u \in_R [0, 2^{2k+2k_2}], a = C_i^{4u}, b = v^u$
#$k = log_2 n_{model}$, k_2 is the security param.
#challenge
2 $c = \mathcal{H}(C_i || C_{i,j} || v || v_j || a || b || pk_{P_j}^{psu})$
#response
3 $\Sigma_{PK}.res(C_i, C_{i,j}, v, v_j; \triangle s_j; a, b; u, c) :=$
$r = u + c \triangle s_j$
4 **return** $Proof_{CD_{i,j}} := (a, b; c; r)$

Algorithm 8. $fsver_2(\Sigma_{CD}; (C_i, C_{i,j}, v, v_j); Proof_{CD_{i,j}}; pk_{P_j}^{psu})$

#verification
#$Proof_{CD_{i,j}} := (a, b; c; r)$
1 $\Sigma_{CD}.ver(C_i, C_{i,j}, v, v_j; a, b; c; r) :=$
$(C_i^{4r} == a(C_{i,j})^{2c}) \bigwedge (v^r == b(v_j)^c)$
2 **return** Yes or No

Under the framework of UVCDN protocol, $\sum_{PK}$ guarantees public auditability if there exist a simulator that can simulate correctness proofs of honest parties, and an extractor that can extract witnesses of corrupted parties which are illustrated by Lemmas 2 and 3, respectively. Similarly, $\sum_{CD}$ also guarantees public auditability shown by Lemmas 4 and 5.

Lemma 2. *Given $X = C(x) = g_{model}^{x} r^{n_{model}}$, $x = \triangle \mathbf{W}$, $r = k_j$, and $c \in \mathbb{C}$ where $\mathbb{C}$ is a finite set called the challenge space, then we have*

$\{d \in_R Z_{n_{model}}; e \in_R Z_{n_{model}}^*; a := g_{model}^{d} e^{n_{model}} X^{-c}: (a; c; d, e)\}$
$\approx$
$\{a_1 \in_R Z_{n_{model}}; b_1 \in_R Z_{n_{model}}^*; a := g_{model}^{a_1} b_1^{n_{model}}; t := (a_1 + cx)/ n_{model}; d := a_1 + cx; e := b_1 k_j^c g_{model}^{t}: (a; c; d, e)\}$

where symbol $\approx$ means that the two distributions are statistically indistinguishable.

Lemma 3. *Let $X = C(x) = g_{model}^{x} r^{n_{model}}$, where $x = \triangle \mathbf{W}$ and $r = k$. Given $(a; s)$ that is generated by the announcement $\Sigma_{PK}.ann$ and two different challenges c, c' with respect to the announcement, there exists an extractor $\mathcal{E}$ that can extract the witness of an adversary $\mathcal{A}$, if $\mathcal{A}$ can present two conversations (d, e) and (d', e') for $(a; s)$, that is,*

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

Lemmas 2 and 3 refer to the property of zero-knowledgeness and soundness in UVCDN protocol's Definition 1, respectively. The concrete proofs can be found in Section 3.2 of [66]. As described in Lemma 2, a simulator without any knowledge of witness of an honest party can provide a proof of encryption correctness, which has statistically indistinguishable distribution compared with a real one. Lemma 3 means that the probability that the extractor $\mathcal{E}$ fails to extract the witness (x, r) of an adversary is negligible, with respect to system security parameter λ.

In terms of $\sum_{CD}$, the corresponding properties of zero-knowledge and soundness for public auditability of correctness decryption are described in Lemmas 4 and 5, respectively. Also, the concrete proofs can refer to Section 4 of reference [67].

Lemma 4. *Given $(C_i, C_{i,j}, v, v_j)$, and $c \in \mathbb{C}$ where $\mathbb{C}$ is a finite set called the challenge space, then we have*

$\{r \in_R [0, 2^{2k+2k_2}]; a = C_i^{4r}(C_{i,j})^{-2c}, b = v^r(v_j)^{-c}: (a, b; c; r)\}$
$\approx$
$\{u \in_R [0, 2^{2k+2k_2}], a = C_i^{4u}, b = v^u; r = u + c \triangle s_j: (a, b; c; r)\}$

where symbol $\approx$ means that the two distributions are statistically indistinguishable.

The above formula means that there exists a simulator without knowledge of $\triangle s_j$ can provide a proof that has a statistically indistinguishable distribution compared with a real one.

Lemma 5. *Given $(C_i, C_{i,j}, v, v_j)$ and $(a, b; u)$ is generated by the announcement $\Sigma_{CD}.ann$. Malicious prover provides two different challenges c, c' with respect to the announcement. There exists an extractor $\mathcal{E}$ that can extract the witness of an adversary $\mathcal{A}$, if $\mathcal{A}$ can present two conversations r and r' for $(a, b; u)$, that is,*

$|1 - \mathbf{Pr}[\mathcal{A}(C_i, C_{i,j}, v, v_j; \triangle s_j; a, b; u; c; c') \to (r; r'); \mathcal{E}(C_i, C_{i,j}, v, v_j; a, b; c; c'; r; r') \to \triangle s_j]| \leq negl(1^{\lambda})$

Lemma 5 shows that extractor $\mathcal{E}$ can extract the witness $\triangle s_j$ of $\mathcal{A}$ with overwhelming probability, with respect to system security parameter λ.

5.3 Fairness Guarantee for Collaborative Training

Recall that we employ two security mechanisms in Blockchain, namely, the trusted time clock mechanism and secure monetary penalty mechanism, to enhance fairness during collaborative training, by following the work [62]. With the trusted time clock mechanism, operations in a contract are forced to finish before the respective time point, as shown in function checkTimeout() in Algorithms 1 and 2. On the other hand, we also define two secure monetary penalty functions for gradient collecting and collaborative decryption, respectively.

In order to prove the property of fairness, Bentov et al. [62] introduced the definition of secure computation with coins (SCC security) in the multi-party setting in a hybrid model that involves not only standard secure computation [68], but also special secure computation dealing with coins. Here, the goal of security refers to fairness presented in their paper. Also, they considered universally composable (UC) security proof for SCC security. In particular, compared to the initial definition of UC security, the view of environment in SCC

TABLE 3
Training Configuration

Parameter	Value
No. of iterations	1500
No. of epochs	1
Learning rate	0.5
Minimal batch size	64

security additionally indicates the distribution of coins because of the added functionality of monetary penalty.

In DeepChain setting, based on the tutorial in Bentov's work [62], the property of fairness for gradient collecting and collaborative decryption is claimed in Section 4.2.4. Our work only replaces the general computation with the special computation to realize functionalities of gradient collecting and collaborative decryption. Other components based on Blockchain, including trusted time clock and monetary penalty exchange, remain unchanged. Thus, the UC-style SCC security defined in Bentov's work can be guaranteed for the specialized functionalities in DeepChain setting, only if SCC security has been proved according to UC composition theorem (refer it to Section 5 of reference [69]). This is demonstrated by Lemma 6, where the environment Z becomes a distinguisher, by following the UC-style proof. If Z with non-uniform probabilistic polynomial-time computation could not distinguish the distribution in the ideal model from that of the hybrid model, then a protocol π SCC realizes a function f.

Lemma 6. *Given an input z, security parameter λ, a distinguisher Z, an ideal process $\texttt{IDEAL}$, an ideal adversary S in $\texttt{IDEAL}$, an ideal function f, and a protocol π that interacts with ideal function g in a model with adversary A, then we have*

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

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

Lemma 7. *Let π be a protocol and f a multiparty function. We say that π securely computes f with penalties if π SCC-realizes the functionality f^*.*

Furthermore, based on Lemma 7 where f is a multiparty function, the security defined for fairness is extended to the multi-party setting as shown in Lemma 6 (as shown by Definition 2 of the work [62]). With protocol π, F is SSC-realized as $F^*_{GradientCollecting}$ and $F^*_{CollaborativeDecryption}$, meaning that they achieve secure gradient collecting and collaborative decryption with penalties, respectively. With these two functionalities and the trusted time clock mechanism, we can guarantee fairness in gradient collecting and collaborative decryption, as shown in Algorithms 3 and 4, respectively.

6 Implementation and Evaluation

In this section, we implement the prototype of DeepChain and evaluate its performance in terms of cipher size, throughput, training accuracy and training time.

6.1 Implementation

We implement building blocks and collaborative training procedures of DeepChain to form three modules, i.e., CordaDeepChain, TrainAlgorithm, and CryptoSystem.

First, we build a Blockchain setting for simulating DeepChain. Blockchain nodes are regarded as parties and workers, and they participate in trading and interact with two pre-defined smart contracts, i.e., *Trading Contract* and *Processing Contract*. Generated transactions are serialized in the Blockchain. Specifically, we use Corda V3.0 [70] to simulate DeepChain for its adaptability and simplification. Corda project is created by R3CEV and has been widely used in banks and financial institutes. It is a decentralized ledger that has some features of Bitcoin and Ethereum [71], such as data sharing based on need-to-know basis and deconflicting transactions with pluggable notaries. A Corda network contains multiple notaries, and our consensus protocol introduced in section 4.2.2 can be executed on them. We build nodes and divide them into parties and workers. Specifically, we set up two CorDapps which agree on Blockchain. The nodes of one CorDapp serve as parties, and the nodes of the other CorDapp play the role of workers. According to the application program interface (API) of Corda, we implement our business logic by integrating three main components, namely, *State*, *Contract* and *Flow*. In particular, an instance of State is used to represent a fact of a kind of data, and it is immutable once an instance of State is known by all nodes at a specific time point. Contract is used to instantiate some rules on transactions. A transaction is considered to be contractually valid if it follows every rule of the contract. An instance of Flow defines a sequence of steps for ledger updates, e.g., how to launch a transaction from a node to another node.

Second, we build the deep learning environment with Python (version 3.6.4), Numpy (version 1.14.0), and Tensorflow (version 1.7.0). We select the popular MNIST dataset [72] which contains 55,000 training samples, 5,000 verification samples and 10,000 test samples. Then, we split randomly this dataset into 10 equi-sized subsets, i.e., each contains $55,000/10 = 5,500$ samples. Then, we conduct multiple training experiments with 4, 5, 6, 7, 8, 9 and 10 parties, denoted as E-4, E-5, E-6, E-7, E-8, E-9 and E-10, respectively. In each experiment, each party possesses one subset of the dataset. Our training model derives from Convolution Neural Network (CNN) with structure: $\texttt{Input} \rightarrow \texttt{Conv} \rightarrow Maxpool \rightarrow \texttt{Fully Connected} \rightarrow \texttt{Output}$. The weights and bias parameters in Conv layer, Fully Connected layer and Output layer 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. We summarize other training parameters in Table 3.

Third, threshold Paillier algorithm is implemented in JAVA. We set the number of bits of modulus n_{model} to 1024 bits, which corresponds to security level of 80 bits. It is worth noting that before executing the encryption algorithm, the weight matrices are assembled into a vector, so that only one cipher is generated for a party.

6.2 Evaluation

We evaluate the feasibility of model training in DeepChain in a multi-party setting by using 4 metrics, that is cipher size, throughput, training accuracy and total cost of time. In particular, we evaluate DeepChain on a desktop computer with 3.3 GHz Intel(R) Xeon(R) CPU and 16 GB memory. Then, for each metric we average the final results over 10

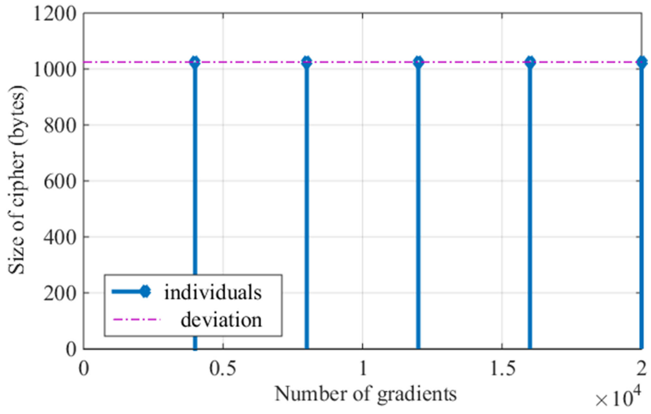

Fig. 4. Impact of No. of gradients on cipher size.

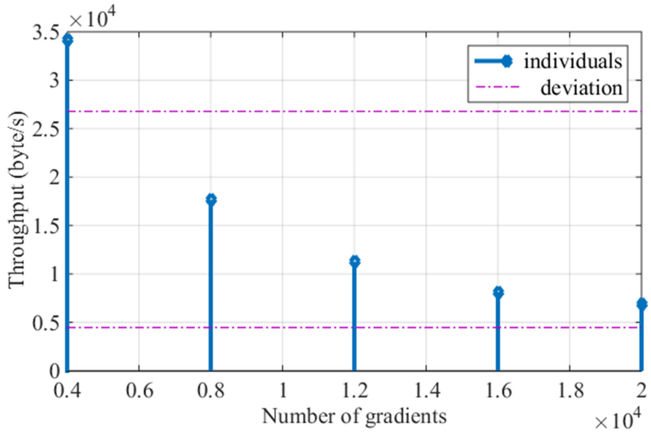

Fig. 5. Impact of No. of gradients on throughput.

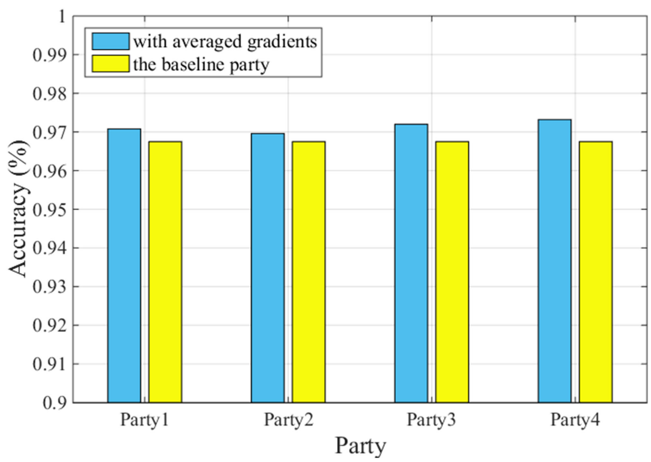

Fig. 6. Training accuracy for the case of four parties.

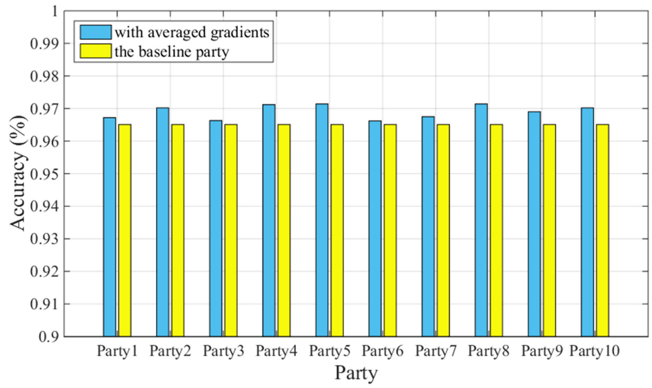

Fig. 7. Training accuracy for the case of ten parties.

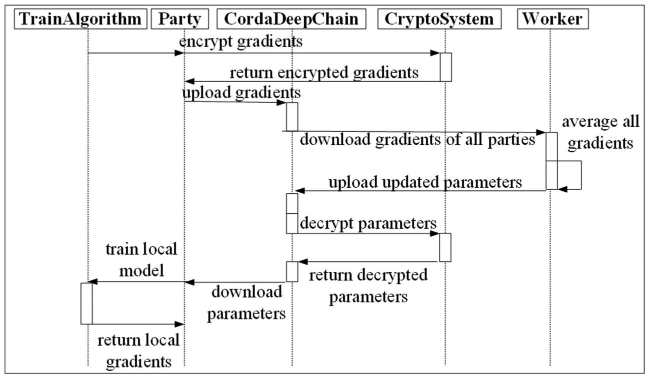

Fig. 8. Interaction in the entire collaborative training process.

trails. As can be seen in Fig. 4, the size of cipher remains constant when we encrypt different amounts of gradients. On the other hand, as the number of gradients increases, the throughput decrease steadily, as shown in Fig. 5.

In terms of training accuracy, we show that the more parties participate in collaborative training, the higher the training accuracy. We create 7 experimental scenarios with 4, 5, 6, 7, 8, 9, and 10 parties, respectively. Each party trains the local model with her training dataset that contains 5,500 samples. Obviously, the more parties in a scenario, the larger the size of the total dataset, for example, the size of the total dataset is 5,500 $\times$ 4 for E-4, and 5,500 $\times$ 10 for E-10.

By sharing gradients in DeepChain, each individual party obtains updated parameters contributed by the gradients from other parties. Specifically, the updated parameters are calculated by taking the average of the gradients of all parties. We also create an external party, denoted as *baseline party*, who only trains the local model on her dataset with 5,500 samples, without taking into account the gradients from other parties. For space limitation, we only give the results for E-4 and E-10.

As shown in Figs. 6 and 7, we can see that for both cases collaborative parties achieve higher training accuracy than the baseline party. Specifically, in Fig. 6 the baseline party has an accuracy of 96.75 percent, whereas Party 1, Party 2, Party 3, and Party 4 achieve 97.08, 96.96, 97.20 and 97.32 percent accuracy, respectively. Similarly, in Fig. 7, the accuracy of the baseline party is 96.51 percent, while the accuracy numbers of Party 1 to Party 10 are 96.72, 97.02, 96.63, 97.12, 97.14, 96.62, 96.75, 97.14, 96.90, 97.02 percent, respectively.

Next, we investigate the time costs of the implemented prototype. Note that we use the same training model on MNIST dataset for each experiment E-4, E-5, E-6, E-7, E-8, E-9 and E-10. The total execution time is the time spent by a party in the entire collaborative training process in DeepChain. We depict the training process in Fig. 8 by using an interaction diagram. It is worth noting that for efficiency, parties only share gradients with 100 times, i.e., every 15 iterations, instead of No. of iterations in each experiment. This assumption follows some researcher's suggestion that

TABLE 4
Interpretations of Time Variants

Variant	Interpretation
$t_{15\_iteration}$	time of 15 iterations
$t_{encrypt}$	time to encrypt gradients
$t_{uploadByParty}$	time to upload gradients
$t_{downloadByWorker}$	time to download gradients from all parties
$t_{average}$	time to average all gradients
$t_{uploadByWorker}$	time to upload updated parameters
$t_{downloadByParty}$	time to download parameters
$t_{decrypt}$	time to decrypt parameters

training accuracy is still acceptable when averaging gradients every 10 to 20 iterations [73].

Let the frequency of sharing gradient be $\#share$, the number of iteration be $\#iteration$, the period of sharing be $\#period \in [10, 20]$ and the number of parties be N. Then, the total time cost can be computed as $Time \approx \frac{\#iteration}{\#period} \times (t_{15\_iteration}) + \#share \times (t_{encrypt} + t_{uploadByParty} + N \times t_{downloadByWorker} + N \times t_{average} + t_{uploadByWorker} + \times t_{downloadByParty} + t_{decrypt})$. For ease of explanation, we summarize all the time-related variables in Table 4.

We assume that in the above formula all the time-related variables achieve their corresponding time costs when $N = 1$. When number of parties N increases, the total time cost also increases slightly, since $t_{downloadByWorker}$ and $t_{average}$ become greater after multiplication. On the other hand, $t_{uploadByParty}$, $t_{uploadByWorker}$ and $t_{downloadByParty}$ grow a little bit with N, but their growth rates are negligible. The reason is that the time used to synchronize State in Corda platform is very short, so that $t_{uploadByParty}$ increases slightly with N. For $t_{uploadByWorker}$ and $t_{downloadByParty}$, no matter how large N is, all parties' ciphers are aggregated into one cipher that is uploaded/downloaded by workers/parties by using relatively constant time. It is worth noting that the time cost is also determined by the size of the training model. Hence, both the number of iterations and the time for cryptographic operations increase with the size of the training model.

From Fig. 9 we can see that the total time cost grows steadily with the number of parties. Specifically, the total time costs for E-4, E-5, E-6, E-7, E-8, E-9, and E-10 are 391,861 s, 392,359 s, 394,533 s, 394,287 s, 395,938 s, 397,252 s and 398,079 s, respectively. The reasons for this trend are that (1) when the number of parties increases, $t_{uploadByParty}$, $t_{downloadByWorker}$, $t_{average}$, $t_{uploadByWorker}$ and $t_{downloadByParty}$ also increase, but the increments are not obvious in the total time, (2) more parties need longer waiting time to synchronize when sharing gradients, and (3) $t_{encrypt}$ and $t_{decrypt}$ dominate the large proportion of the total time cost that depends on the size of the training model instead of on the number of parties. In addition, based on our experiments it is expected that when the number of parties is more than 1306, the total time would increase significantly with N, because the time $N \times (t_{downloadByWorker} + t_{average})$ is greater than time $(t_{encrypt} + t_{decrypt})$.

7 Conclusion and Future Work

In this paper we present DeepChain, a decentralized framework based on Blockchain for secure collaborative deep training. We introduce an incentive mechanism and achieve three security goals, namely confidentiality, auditability, and fairness. Specifically, we formalize the incentive mechanism based on Blockchain, which possesses compatibility and liveness properties. Through our incentive mechanism we demonstrate that participants are incentivized to behave correctly with high probability.

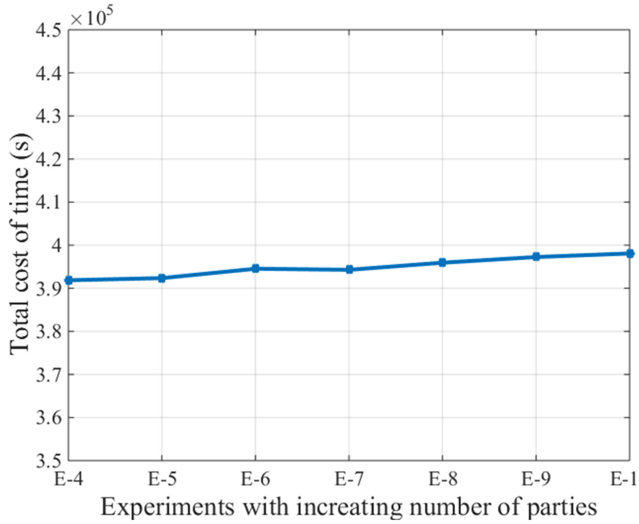

Fig. 9. Total time cost with increasing number of parties.

For confidentiality of local gradients, we employ Threshold Paillier algorithm. In particular, by combining a carefully designed component into the encryption algorithm, we achieve the goal that only one cipher is generated for a party. In addition to confidentiality, our DeepChain provides auditability and fairness. We use non-interactive zero-knowledge to prove auditability of the collaborative training process. We also design timeout-checking and monetary penalty mechanisms to guarantee fairness among the participants. We implement a prototype of DeepChain and evaluate it on real dataset, with respect to cipher size, throughput, training accuracy and training time.

Our DeepChain may also have some impact on model-based pricing market. Since DeepChain stores not only the training parameters, but also the trained models, they can be used for paid deep learning services when the model-based pricing market is mature. Participants who possess the trained models may have long-term financial benefits, since they can provide deep learning services to those who are unable to build the model by themselves but willing to pay for the services. On the other hand, all training processes and the model parameters are recorded, which could be used in transfer learning. For example, some information of the trained models could be re-used to train a new similar model. Of course, for transfer learning case, the security problems should be re-defined and analyzed, which are left in our future work.

Acknowledgments

Jian Weng was supported by National Natural Science Foundation of China (Grant Nos. 61825203, U1736203, and No. 61732021), Guangdong Provincial Special Funds for Applied Technology Research and Development and Transformation of Key Scientific and Technological Achievements (Grant

No. 2016B010124009), and Science and Technology Program of Guangzhou of China (Grant No. 201802010061), and Guangdong Provincial Basic and Applied Research Major Programme (Grant No. 2019B030302008). Jiasi Weng was supported by National Key R&D Program of China (Grant No. 2018YFB1402600). Jilian Zhang was supported by National Key R&D program of China (Grant No. 2018YFB1003701), National Natural Science Foundation of China (Grant No. 61972177) and Communication and Computer Network Lab of Guangdong (Grant No. CCNL201903). Ming Li was supported by National Key Research and Development Plan of China (Grant No. 2017YFB0802203), and Graduate School of Jinan University. Yue Zhang was supported by National Natural Science Foundation of China (Grant No. 61872153). Weiqi Luo was supported by National Natural Science Foundation of Chain (Grant No. 61877029), Guangdong Provincial Special Funds for Applied Technology Research and Development and Transformation of Key Scientific and Technological Achievements (Grant No. 2017B010124002).

References

[1] G. Hinton et al., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 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 Trans. Image Process.*, vol. 24, no. 12, pp. 5017–5032, Dec. 2015.

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

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

[5] P. Danaee, R. Ghaeini, and D. A. Hendrix, "A deep learning approach for cancer detection and relevant gene identification," in *Proc. Pacific Symp. Biocomputing*, 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 *Proc. IEEE 16th Int. Conf. Data Mining*, 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 *Proc. Usenix Conf. Operating Syst. Des. Implementation*, 2016, pp. 571–582.

[8] T. Chen and S. Zhong, "Privacy-preserving backpropagation neural network learning," *IEEE Trans. Neural Netw.*, vol. 20, no. 10, 2009, Art. no. 1554.

[9] A. Bansal, T. Chen, and S. Zhong, "Privacy preserving back-propagation neural network learning over arbitrarily partitioned data," *Neural Comput. Appl.*, 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 Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 212–221, Jan. 2014.

[11] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. Allerton Conf. Commun., Control, Comput.*, 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 Comput.*, 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 Trans. Comput.*, vol. 65, no. 5, pp. 1351–1362, May 2016.

[14] K. Bonawitz et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 1175–1191.

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

[16] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1333–1345, May 2018.

[17] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 587–601.

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

[19] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the gan: information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 603–618.

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

[21] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1333–1345, May 2018.

[22] A. Pyrgelis, C. Troncoso, and E. D. Cristofaro, "Knock knock, who's there? membership inference on aggregate location data," in *Proc. 25th Annu. Netw. Distrib. Syst. Security Symp., NDSS 2018*. The Internet Society, 2018, pp. 1–15.

[23] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," 2018, *CoRR*, vol. abs/1807.00459, 2018.

[24] "Health insurance portability and accountability act." 1996. [Online]. Available: https://www.hhs.gov/hipaa/index.html

[25] J. Vaidya, B. Shafiq, X. Jiang, and L. Ohno-Machado, "Identifying inference attacks against healthcare data repositories," *AMIA Summits Translational Sci. Proc.*, vol. 2013, 2013, Art. no. 262.

[26] G. Heigold et al., "Multilingual acoustic models using distributed deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2013, pp. 8619–8623.

[27] R. Jurca and B. Faltings, "An incentive compatible reputation mechanism," in *Proc. EEE Int. Conf. E-Commerce*, 2003, pp. 285–292.

[28] U. Shevade, H. H. Song, L. Qiu, and Y. Zhang, "Incentive-aware routing in dtns," in *Proc. IEEE Int. Conf. Netw. Protocols*, 2008, pp. 238–247.

[29] S. Zhong, J. Chen, and Y. R. Yang, "Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks," in *IEEE INFOCOM 2003. Proc. 22nd Annu. Joint Conf. IEEE Comput. Commun. Societies*, vol. 3, 2003, pp. 1987–1997.

[30] B. B. Chen and M. C. Chan, "MobiCent: A credit-based incentive system for disruption tolerant network," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[31] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: http: //bitcoin.org/bitcoin.pdf

[32] M. Ben-Or and A. Hassidim, "Fast quantum byzantine agreement," in *Proc. 37th Annu. ACM Symp. Theory Comput.*, 2005, pp. 481–485.

[33] S. Micali, "ALGORAND: The efficient and democratic ledger," *CoRR*, vol. abs/1607.01341, 2016.

[34] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Operating Syst. Princ.*, 2017, pp. 51–68.

[35] K. Nikitin et al., "CHAINIAC: Proactive software-update transparency via collectively signed skipchains and verified builds," in *Proc. 26th USENIX Security Symp.*, 2017, pp. 1271–1287.

[36] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 792–800.

[37] Y. Zhang, C. Xu, J. Ni, H. Li, and X. S. Shen, "Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, p. 1, 2019, doi: 10.1109/TCC.2019.2923222.

[38] A. B. Kurtulmus and K. Daniel, "Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain," *CoRR*, vol. abs/1802.10185, 2018.

[39] E. B. Sasson et al., "Zerocash: Decentralized anonymous payments from bitcoin," in *Proc. IEEE Symp. Security Privacy*, 2014, pp. 459–474.

[40] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *Proc. IEEE Symp. Security Privacy*, 2013, pp. 397–411.

[41] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Security Privacy*, 2016, pp. 839–858.

[42] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Englewood Cliffs, NJ, USA: Prentice Hall PTR, 1994.

[43] H. Cui, G. R. Ganger, and P. B. Gibbons, "Scalable deep learning on distributed GPUs with a GPU-specialized parameter server," in *Proc. Eleventh Eur. Conf. Comput. Syst.*, ACM, 2016, pp. 1–16.

[44] H. Ma, F. Mao, and G. W. Taylor, "Theano-MPI: A theano-based distributed training framework," in *Proc. Eur. Conf. Parallel Process.*, 2016, pp. 800–813.

[45] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *Proc. 2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, 2017, pp. 181–193.
[46] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, and S. Pollin, "Distributed deep learning models for wireless signal classification with low-cost spectrum sensors," *IEEE Trans. Cogn. Commun. Netw.*, vol. 4, no. 3, pp. 433–445, 2018.
[47] C. Hardy, E. Le Merrer, and B. Sericola, "Distributed deep learning on edge-devices: feasibility via adaptive compression," in *Proc. IEEE 16th Int. Symp. Netw. Comput. Appl. (NCA)*, 2017, pp. 1–8.
[48] J. Dean et al., "Large scale distributed deep networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.
[49] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with FBFFT: A gpu performance evaluation," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015, pp. 1–17.
[50] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," *CoRR*, vol. abs/1501.02876. 2015.
[51] M. Lin, S. Li, X. Luo, and S. Yan, "Purine: A bi-graph based deep learning framework," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015, pp. 1–6.
[52] L. Chen, P. Koutris, and A. Kumar, "Towards model-based pricing for machine learning in a data marketplace," in *Proc. Int. Conf. Manag. Data, SIGMOD*, ACM, 2019, pp. 1535–1552.
[53] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya, "Incentivizing outsourced computation," in *Proc. 3rd Int. Workshop Economics Networked Syst.*, 2008, pp. 85–90.
[54] A. Küpçü, "Incentivized outsourced computation resistant to malicious contractors," *IEEE Trans. Dependable Secure Comput.*, vol. 14, no. 6, pp. 633–649, Nov./Dec. 2017.
[55] A. Demers et al., "Epidemic algorithms for replicated database maintenance," in *Proc. 6th Annu. ACM Symp. Princ. Distrib. Comput.*, 1987, pp. 1–12.
[56] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," PhD dissertation, The Univ. Guelph, Guelph, ON, Canada, 2016.
[57] J.-S. Weng, J. Weng, M. Li, Y. Zhang, and W. Luo, "Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive," Cryptology ePrint Archive, Rep. no. 2018/679, 2018, . [Online]. Available: https://eprint.iacr.org/2018/679
[58] F. McKeen et al., "Innovative instructions and software model for isolated execution," *Proc. 2nd Int. Workshop Hardware Architectural Support Security Privacy*, 2013, Art. no. 10.
[59] P.-A. Fouque, G. Poupard, and J. Stern, "Sharing decryption in the context of voting or lotteries," in *Proc. Int. Conf. Financial Cryptography*, 2000, pp. 90–104.
[60] T. Nishide and K. Sakurai, "Distributed paillier cryptosystem without trusted dealer," in *Proc. Int. Workshop Inf. Security Appl.*, 2010, pp. 44–60.
[61] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[62] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Proc. Int. Cryptology Conf.*, 2014, pp. 421–439.
[63] R. Kumaresan and I. Bentov, "How to use bitcoin to incentivize correct computations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2014, pp. 30–41.
[64] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 1999, pp. 223–238.
[65] B. Schoenmakers and M. Veeningen, "Universally verifiable multiparty computation from threshold homomorphic cryptosystems," in *Proc. Int. Conf. Appl. Cryptography Netw. Security*, 2015, pp. 3–22.
[66] I. B. Damgård and M. J. Jurik, "Efficient protocols based on probabilistic encryption using composite degree residue classes," *BRICS Rep. Ser.*, vol. 7, no. 5, pp. 1–8, 2000.
[67] V. Shoup, "Practical threshold signatures," in *Proc. 19th Int. Conf. Theory Appl. Cryptographic Techn.*, 2000, pp. 207–220.
[68] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[69] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2001, pp. 136–145.
[70] "Corda: An open source distributed ledger platform." 2016. [Online]. Available: https://docs.corda.net/
[71] G. Wood et al., "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.
[72] C. J. B. Y. LeCun and C. Cortes, "The MNIST database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/
[73] H. Su and H. Chen, "Experiments on parallel training of deep neural network using model averaging," *CoRR*, vol. abs/1507.01239. 2015.

Jiasi Weng received the BS degree in software engineering from South China Agriculture University, in June 2016. Currently, she is working toward the PhD degree in the College of Information Science and Technology & College of Cyber Security, Jinan University. Her research interests include applied cryptography, blockchain and privacy.

Jian Weng received the BS and MS degrees from South China University of Technology, in 2001 and 2004, respectively, and the PhD degree from Shanghai Jiao Tong University, in 2008. He is a professor and an executive dean with the College of Information Science and Technology, Jinan University. His research interests include public key cryptography, cloud security, blockchain, etc. He has published 80 papers in international conferences and journals such as CRYPTO, EUROCRYPT, ASIACRYPT, TCC, PKC, CT-RSA, the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Transactions on Dependable and Secure Computing*, etc. He also serves as associate editor for the *IEEE Transactions on Vehicular Technology*. He received the Young Scientists Fund from the National Natural Science Foundation of China in 2018, and the Cryptography Innovation Award from Chinese Association for Cryptologic Research (CACR) in 2015. He served as general co-chair for SecureComm 2016, TPC co-chairs for RFIDsec'13 Asia and ISPEC 2011, and program committee members for more than 40 international cryptography and information security conferences. He is a member of the IEEE.

Jilian Zhang received the MS degree from Guangxi Normal University, China, in 2006, and the PhD degree from Singapore Management University, in 2014. He is currently an associate professor with the College of Cyber Security, Jinan University, Guangzhou China. His research interests include data management, query processing, and database security. His work has been published on international journals and conferences, including the *IEEE Transactions on Knowledge and Data Engineering*, ACM SIGMOD, VLDB, and IJCAI.

Ming Li received the BS degree in electronic information engineering from the University of South China, in 2009, and the MS degree in information processing from Northwestern Polytechnical University, in 2012. From 2016, he is working toward the PhD degree with the Colledge of Information Science and Technology & College of Cyber Security, Jinan University. His research interests include crowdsourcing, blockchain and its privacy and security.

Yue Zhang is working toward the PhD degree with the Colledge of Information Science and Technology & College of Cyber Security, Jinan University. His research interests include system security, especially IoT security. He has published papers in international conference and journals, such as the *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Vehicular Technology*, RAID, etc.

Weiqi Luo received the BS and MS degrees from Jinan University, in 1982 and 1985, respectively, and the PhD degree from South China University of Technology, in 1999. Currently, he is a professor with the School of Information Science and Technology, Jinan University. His research interests include network security, big data, artificial intelligence, etc. He has published more than 100 high-quality papers in international journals and conferences.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.