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

Jia-Si Weng, Jian Weng, *Member, IEEE,* Ming Li, Yue Zhang, Weiqi Luo,
E-mail: cryptjweng@gmail.com

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

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

## 1 Introduction

Recent advances in deep learning based on artificial neural networks have performed unprecedented accuracy in various tasks, e.g., speech recognition, image recognition and other attractive domains such as drug discovery and gene analysis related to cancer [1]. For gaining the higher accuracy, deep learning models to be trained can be more complex, and need numerous data to feed, which consume both massive computation and time [2], [3]. This issue of massive consuming can be migrated by employing distributed deep learning technology which also is widely researched in recent years. These developments promote deep learning technologies obtaining more excellent achievements, and also lead to the privacy issue of deep learning receiving much concern. Thus, many researchers pay great attention to the topic of privacy-preserving deep learning.

A few work has performed their methods for privacy-preserving deep learning in different system models [4]–[9]. Among these work, the methods in the setting of federated learning are widely discussed. Federated learning [1] essentially is derived from the technology of distributed deep learning where training data is partitioned to multiple parties. Multiple parties locally train the same model on local data and upload training intermediate gradients to the parameter server maintaining the model. The parameter server aggregates gradients from multiple parties and updates the parameters of the model. Then parties download the updated parameters to the local model and continue to train it. The training procedure goes on iteratively and ends when the training errors occur below pre-set thresholds. Actually, collaborative training by the federal-based way avoids the exposure of training data, but it cannot really protect privacy of training data. There exists a series of work indicating that training intermediate gradients can be used to infer significant information about training data [10]. Shokri *et. al* [7] applied differential privacy technique adding a spot of noises to uploading gradients, which demonstrated a trade-off between data privacy and training performance. However, Hitaj *et. al* [11] pointed out that their work failed to protect data privacy and indicated a curious parameter server can learn private data by GAN (general adversary network) learning. Phong *et. al* [12] also concerned privacy threats of data from a curious parameter server, and protected data privacy by utilizing the homomorphic encryption technique against the learning of the server. Nevertheless, their scheme assumed that collaborative parties are honest but not curious; that cannot prevent some party's gradients or data from being exposed to a curious party. The proposed method by Bonawitz *et. al* [13] can resist this threat, which employed secret sharing and symmetric encryption mechanism to ensure the confidentiality of parties' gradients. The resistance to be effective needs some assumptions that parties and parameter server can not collude to a certain, and aggregated gradients in the clear text reveal nothing

• *J. S. Weng, J. Weng, M. Li, Y. Zhang and W. Luo are with the College of Information Science and Technology in Jinan University, and Guangdong/Guangzhou Key Laboratory of Data Security and Privacy Preserving, Guangzhou 510632, China.*
*Jian Weng is the corresponding author.*

1. It is also called as collaborative learning, and we do not distinguish them in the paper.

about parties' data. The second assumption may need to be re-concerned, since a membership inference attack on aggregated location data is newly proposed [14]. The aforementioned work generally prevented the privacy threat from a curious parameter server. It is actually not enough to simply consider such a parameter server because its other malicious behaviors can disrupt the whole procedure of collaborative training, such as dishonestly collecting gradients or updating parameters. From this view, a securer method is desirable that no only guarantees the confidentiality of gradients but also ensures the computation verifiability of gradient collecting and parameter updating.

On the other hand, those schemes are presented in a common situation where there exist multi-parties possessing data and being prepared to accomplish training. But even data sparseness does be a bottleneck to obtain a well-trained deep learning model in general [15], especially to obtain a large, complex and high-accuracy model. For examples, in medical research, normal samples always are far more than acromegalic samples. Individual institutes may only possess small quantities of data which is not enough to train a good model. Many promising companies and research institutes focusing on deep learning techniques also devote a lot to collect enough data [7]. Nevertheless, benefit and privacy considerations make it difficult to be accomplished. Available data means unlimited value and is linkable and inferrable which may disclose more hidden private information. In this case, it may be feasible to build up an incentive setting where distrustful data-possessing individuals, even if they only possess a piece of data, are value-driven to participate in collaborative training. Meanwhile, gradients collecting and parameter updating are incentivizable to be accomplished honestly. It is expected to eventually achieve win-win benefit, in which collaborative parties can gain a well-trained deep learning model and solve corresponding AI tasks.

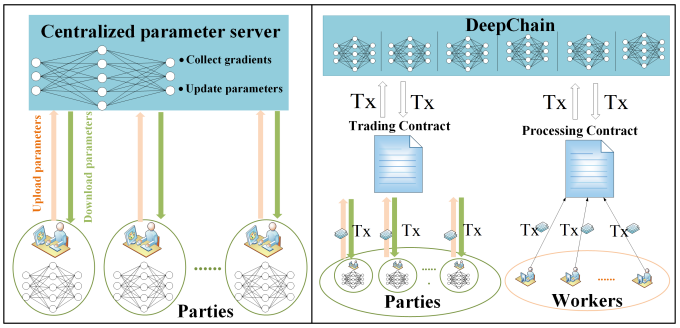


Fig. 1. The system model comparison between traditional training and DeepChain (Tx means transactions covering the entire collaborative training; Trading Contract and Processing Contract are Smart Contract on DeepChain, which together instructs the secure training.)

With the aforementioned consideration, we notice it is feasible to find a solution by integrating an exquisitely Blockchain-based incentive with pro-developing cryptographic primitives. Therefore, we build DeepChain, a healthy and win-win decentralized platform based on Blockchain for privacy-preserving deep learning training. The system models of traditional training and DeepChain are presented in Fig. 1. It makes massive locally training intermediate gradients be securely aggregated among distrustful owners via launching transactions. Parameter updating is incentivizedly computed by workers via processing transactions, thereby accomplishing iterative training. In addition, data ownerships and the entire training process are publicly auditable. In conclusion, we make our contributions as follows.

- DeepChain recognizes the value of separated data pieces, and provides an reliable incentive mechanism to aggregate them for collaborative training.
- It provides the auditability and confidentiality for locally training gradients of each participant.
- It in addition employs economic affair to push each participant behaving honestly, which the fairness for collaboration is guaranteed.
- By introducing the promising security into the incentive mechanism, gradient collecting and parameter updating are incentivized to be honestly accomplished.
- DeepChain also brings long-term benefits. All training processes and well-trained models are recorded, which could advance the development of transfer learning.

The rest of the paper is organized as follows. In Section 2, we summary the existing related work. In Section 3, we give a brief introduction on Blockchain and the background of deep learning. Then, we describe the threat model and security requirements in Section 4. In Section 5, we present an overview and further introduce concrete design details of DeepChain. We analyze security properties of DeepChain in Section 6 and in the Section 7 we display the implementation of DeepChain and evaluation results. Lastly, we conclude the paper in Section 8.

## 2 Related Work

This work is supported by the related works twofold. In this section, the topic about privacy-preserving training is given at first and then we discuss the works on distributed deep learning.

### 2.1 Privacy-preserving training

There are mainly two kinds of works on privacy-preserving neural network learning, one of which is privacy-preserving training and other one is privacy-preserving classification. This paper falls in the area of the former, so we only display here the related works of privacy-preserving training. Chen *et. al* [4] proposed privacy-preserving two-party algorithm of multi-layer neural networks by using homonorphic encryption. However, they focus on party data which is vertically partitioned. Bansal *et. al* [5] first considered arbitrarily partitioned data between two parties for privacy-preserving back-propagation neural network learning. Their algorithm guarantees the privacy of each party' training data, in which two party only share the random values of weights after each round of training and learn the result weights just at the end of training. The above two works only focus on the two-party setting. Yuan *et. al* [6] introduced the power of cloud computing to the multi-party setting and proposed

privacy-preserving back-propagation algorithm on arbitrarily data. Their scheme allowed multiple parties to upload locally encrypted data to the cloud and the cloud undertake to execute learning algorithm on encrypted training data. With the popularity of big data feature by employing deep learning technique, Zhang *et. al* [9] presented effective privacy-preserving deep learning model by migrating the complex operations to the cloud. Their method successfully used the currently most efficient BGV encryption scheme to support full homomorphic operations and avoid computing exponentiation operation by approximating the activation function as a polynomial function. Following the works on privacy-preserving training with the power of the cloud, Li *et. al* [8] discussed those works only allow individual training data to be encrypted with the same public key. They claimed this kind of method is limited to the scenario where data are from different parties who encrypted data with different public key. Thus, they took it into consideration and proposed a cloud computing-based framework to privacy-preserving training by utilizing a proxy fully homomorphic encryption scheme. Shokri *et. al* [7] implemented a privacy-preserving deep learning system where multiple parties share a small fraction of gradients and learn a deep learning model together. Their system assume parties are willing and honest to upload their gradients to a centralized server and the server always maintains the latest training results for being downloaded by parties. They did not consider the necessary to audit the behaviors of joint parties and the server, and provide the fairness among them. Their work also did not achieve the unlinkability of parties' sharing parameters although they recognized the importance of it. Considering about training on communication-expensive and instable mobile devices, Bonawitz *et. al* [13] preserved Shokri *et. al*'s [7] system model of training and proposed an efficient method to securely aggregate parties's local gradients for a common deep learning model. Different from a single server model of above shemes, Mohassel *et. al* [16] explored the method of the two-server model, and constructed SecureML where two non-colluding servers collect and accomplish the training while preserving the privacy of data.

### 2.2 distributed deep learning

Modern deep learning model trained on massive datasets requires significant amount of computation power and extremely long time-consuming, so distributed deep learning training have been condcted in recent years. There are two approaches to distributed deep learning training: model parallelism and data parallelism, the difference of which is the partitioning on a total model or dataset. Actually, distributed training can work is due to most of modern deep learning algorithms are based on stochastic gradient descent (SGD) which can execute parallizedly and asynchronously [17]–[21]. DistBelief was proposed to enable training large-scale DNNs which supported both model and data partitioning. However, it cost significant amounts of computation (2000 machines for the 22K ImageNet classification task) [19]. Another attractive aychronously distributed system was implemented by Adam with model partitioning. Adam demonstrated more efficient and scalable training than DistBelief (120 machines for the same task with DistBelief) [3]. SINGA also was proposed based on model partitioning, but it allowed both synchronous and asynchronous training [22]. Yan *et. al* considered the impact of model and data partitioning on training large DNNs. By an optimizer on the system scalability, they finally gained the best configurations which achieved high accuracy and minimized the training time [23]. Recently, a series of works on distributed deep learning training are continuely proposed [24]–[28], which showed us the feasibility to research on collaborative training a deep learning model.

## 3 Background

### 3.1 Blockchain technology

Blockchain technology has arisen a surge of interests both in the research community and industry. It becomes an emerging technology as a decentralized, immutable, sharing and time-order ledger. Transaction are stored into blocks containing timestamps and references (i.e., the hash of a previous block) which are maintained as a chain. In Blockchain, transactions are created by pseudonymous participants and competitively collected to build a new block by a role named worker. The worker of a new and valid block can gain amount of rewards so that the chain is continuously lengthened by competitive workers. That presents the incentive mechanism in the Blockchain setting. In addition, pro-developing Blockchain technologies introducing smart contract support Turing-complete programmability, such as Ethereum and Hyperledger. On the other hand, a series of works on transaction privacy are popular by applying cryptographic tools into Blockchain, such as Zerocash [29], Zerocoin [30] and Hawk [31]. Therefore, Blockchain technology's incentive feature and its pro-developing technologies inspire us to solve our problem.

### 3.2 Deep learning training

The backbone of a deep learning model consists of three layers: the input layer, the hidden layer and the output layer. A deep learning model can contain multiple hidden layers which represents the depth of a model. Each layer includes amount of neurones. Neurones in different layers enable to learn hierarchical features from training data and then obtain feature representations in different levels of abstraction. Each neurone has multiple inputs and a single output. Typically, the outputs of neuron $i$ in layer $l-1$ connects to the inputs of all neurones in layer $l$. Each connection is associated with a weight $w$. For example, $w_{j,i}$ associates the connection between neuron $j$ in layer $l-1$ and neuron $i$ in layer $l$. Each neuron $i$ also has a bias $b_i$. These weights and bias are called as model parameters which need to be learned via training.

The currently popular deep model learning method is Back-Propagation (BP) learning algorithm which is composed of two steps: feed forward and back-propagation. Specifically, in the step of feed forward, the values in each layer are calculated based on the parameters of the preceding layer and the current-layer parameters. A key component in the calculation is called as activation which is the output of each neuron $i$. Activation is used to learn nonlinear features of inputs via a function $Act(\cdot)$. For computing

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

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

## 4 Threats and Security Goals

In this section, we discuss twofold threats in the scenario where distrustful parties share locally training gradients for collaborative training. Then, we present which kinds of security properties provided by DeepChain with respected to the threats.

**Threat 1: Disclosure of data and model.** This threat refers to three hands: 1) the disclosure of any individual's updated gradients enables adversaries to infer respective and available information about local data and model. Adversaries may initiate inference attacks or membership attacks [36]. 2) a collaborative training model without protection may leak private information. Adversaries may launch parameter inferring attacks, thus breaking the privacy for participants [11]. 3) the leakage of aggregating updated gradients may lead to some linkability threats [14].

**Security Goal for Threat 1: Confidentiality guarantees for training gradients.** Note that DeepChain creates a platform trading gradients for participants to collaboratively train a significant model. Each participant individually encrypts and trades training gradients belonging to his local model based on personal data. All gradients then are used to update the parameters of collaborative training model which also is collaboratively encrypted by participants. In each iteration, participants obtain updated parameters via collaboratively decryption. Assume that participants themselves do not expose data, and a threshold $t$ of participants are honest in the entire produce, which means no more than $t$ participants colluding to disclose parameters. Individuals' training gradients do not be exposed to anyone unless $t$ participants collude, from which the first kind of threat is resisted. In terms of the second threat, only if participants do not disclose the updated parameters of the collaborative model he gain, any external parity could not gain any information.

**Threat 2: Internal disruption for collaborative training.** Considering the situation that participants may be malicious, and their disruption behavior may lead to a bad collaborative training. They may at random choose their inputs instead of a correctly encrypted construction for gradients, which disrupts the evaluation operations on ciphers, obstructing the training. The evaluation results on encrypted gradients for parameter updating may be computed as an incorrect result. During the phase of collaborative decryption, participants may give a problematic decryption share. They may be selfish, and then even independently abort the training. The aforementioned malicious behaviors by disrupted participants can block up the training, as a result that honest participants are unable to gain a good trained model.

**Security Goal for Threat 2: Public auditability for gradient collecting and parameter updating.** Assume that the majority of participants for gradient collecting are honest and more than $2/3$ workers are honest to update parameters on DeepChain. In terms of gradient collecting, participants' transactions containing encrypted gradients additionally provide proofs for correctness, which allows the public to audit whether some participant gives a correctly encrypted construction for gradients or not. On the other hand, workers claim computation results for parameter updating via transactions which are recorded on DeepChain. Those transactions are publicly auditable, and computation results presented by them are convincing to be correct only if $2/3$ workers are honest. With parameter updating, updated parameters are to be collaboratively decrypted where participants provide their decryption shares and corresponding proofs for correctness. Similar to the case of gradient collecting, anyone can audit whether the decryption shares are correct or not.

**Security Goal for Threat 2: Fairness enhancement for collaborative training.** Fairness is enhanced by a trusted timeout-checking and a monetary penalty mechanisms on DeepChain. This refers to the consistence of participants' behavior and benefit by defining time clocks for each functions within smart contracts for accomplishing a collaborative training. Specifically, at a time point following a function, the accomplished results of this function are verified. When the function is verified to be failed, it represents two cases where participants behave dishonestly. One case is that there exists participants not being punctual at the time

point. Another case means that there exists participants who incorrectly accomplish the function. In this two cases, the monetary penalty mechanism works by revoking the pre-frozen deposit of a dishonest participant, and allocating them to honest participants. As a result, fairness ensures honest participant who both behave punctually and correctly must not be punished, and honest participant will be compensated if there exists some participant behaving dishonestly.

## 5 System Design

In this section, we give the overview of collaborative training on DeepChain as shown in Fig. 2 and introduce its concrete design details.

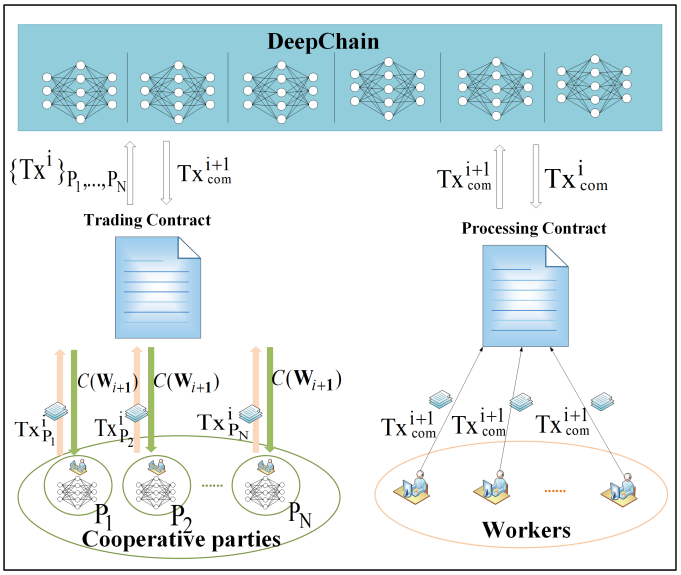


Fig. 2. The overview of collaborative training on DeepChain

### 5.1 Overview

Informally, DeepChain integrates special features of Blockchain and cryptographic tools to provide a platform which publicly recognizes the value of private data owned by individual parties. We assume the condition of network communication is synchronous among parties. Suppose that parties $P_j (j \in \{1, ..., N\})$ constitute a cooperative group for collaborative training. They agree on pre-defined information for training including to initialize a collaborative model to be trained. Those pre-defined information is generated into a transaction $Tx^0_{com}$ by signing with cooperative parties. Suppose that an address attaching to the transaction is $pk_{it_0}$ where $it_0$ means the beginning of iterative training. At the end of each iteration $i$, the updated model in $Tx^i_{com}$ is attached to a new address $pk_{it_i}$ known to cooperative parties. Intermediate gradients from cooperative parties via transactions $Tx^i_{P_j}$ are collected by *Trading Contract* during each iteration. Those intermediate gradients are their locally training weights $C_{P_j}(\triangle \mathbf{W}_{i,j})$ where $C$ means the corresponding ciphers and $i$ represents the iteration index of training. When accomplishing the collection of transactions, *Trading Contract* uploads transactions to DeepChain $\{Tx^i\}_{P_1,...,P_N}$. Workers perform their contributions by processing transactions $\{Tx^i\}_{P_1,...,P_N}$ via *Processing Contract*. They update the weights of the latest model $C(\mathbf{W}_i)$ in $Tx^i_{com}$ into $C(\mathbf{W}_{i+1})$ in $Tx^{i+1}_{com}$ with $C(\triangle \mathbf{W}_{i,j})$ by computing $C(\mathbf{W}_{i+1}) = C(\mathbf{W}_i) \cdot 1/N \cdot \prod_{j=1}^{N} C_{P_j}(\triangle \mathbf{W}_{i,j})$. Due to the incentive mechanism on DeepChain, workers are competitive to perform for monetary rewards. Specifically, workers process transactions competing for a leader who would be rewarded by DeepChain. DeepChain maintains the payoff maximization of parties and workers, so that a healthy and win-win environment is created. This can refer to section 5.2.2. In order to give a better understanding of DeepChain, we firstly introduce the materia terms and several processes to accomplish model training. We further describe the foundational significance of DeepChain. The material terms are as following:

- **Party** DeepChain attracts those parties who desire but fail to accomplish their AI tasks due to limited available data, to cooperate with others having the similar tasks. Parties display their assert (i.e., data) and the brief descriptions of their tasks related to asserts when they first registers DeepChain. All task descriptions of registered parties are public on DeepChain.
- **Trading** A trading is generated by a pseudorandom $pk$ of some cooperative party and sent to an appointed cooperative $pk$ referred to the same goal. Here trading data are locally training weights but not raw data. A trading also is built by a worker processing a group of transactions where worker's $pk$ means the sender of the trading.
- **Cooperative group** A cooperative group is set up by the parties who have similar AI tasks. In this group, they collaboratively trade their locally training weights for the same goal, i.e., cooperative training model to be referred.
- **Cooperative training model** When a cooperative group is set up, they agree on a collaborative training model to be trained. This step includes defining initial model and initializing model parameters. In the following iterations, the cooperative parties contribute to this commonly maintained model by trading their locally training weights. Those trading weights among cooperative parties are used to update their collaborative training model.
- **Locally training model** Locally training model is locally trained by each party. At the end of a local iteration, a party generates a trading attaching his local weights to the appointed $pk$ which is called *Trading Contract*.
- **Worker** Workers are incentive to process transactions that include individual training weights for the collaborative model updating by $C(\mathbf{W}_{i+1}) = C(\mathbf{W}_i) \cdot 1/N \cdot \prod_{j=1}^{N} C_{P_j}(\triangle \mathbf{W}_{i,j})$. They compete to work out a block as a leader so that they gain block rewards and good reputation. Gained rewards and reputation would be evaluated when the worker exchanges them with trained models on DeepChain for accomplishing his AI task.
- **Iteration** An iteration is referred to a processing unit

to train a deep learning model. After an iteration, parameters in a model are updated one time. To train a deep learning model needs multiples iterations. A cooperative group agree on the iteration times before training their model.

- **Round** An round relates to the time interval of a block generation on DeepChain. In a round, transactions from different cooperative groups are processed, which demonstrates the updated results on the corresponding models.
- **DeepCoin** DeepCoin $\$Coin$ is a kind of value representation on DeepChain. Users participate in DeepCoin circulation by acting on DeepChain. In particular, a trained cooperative model brings participants with $\$Coin$ due to each newly generating block on DeepChain. We note that participants consist of two roles: party and worker. Parties in a cooperative group gain $\$Coin$ due to their contributions on training parameters. Workers are rewarded with $\$Coin$ according to their behaviors on DeepChain where they help to update training models corresponding to cooperative groups. Moreover, well-trained models being used need $\$Coin$. Those who have not AI tasks and meanwhile fail to get trained models spend $\$Coin$ to use available models. This can be related with serval recent work on model-based pricing for machine learning [37], [38]. In addition, we add a property called a validity value to DeepCoin. The size of the validity value is set as an interval of a round. The new introduction of the validity value is related to the consensus mechanism played on DeepChain. We will further describe it in 5.2.5.

Then, the procedure to accomplish a collaborative model training includes (1) **cooperative group establishment** where parties who desire to gain a similar model build up a cooperative group according to the similarity of their individual possessing data; (2) **collaborative information commitment** where cooperative parties agree on necessary collaborative information for training and security commitment for security requirement; (3) **gradient collecting via trading contract** where cooperative parties securely upload their locally iterative parameters to a trading contract; (4) **parameter updating via processing contract** where locally iterative parameters from individual parties are contributed to a common model by workers via *Processing Contract*, so that an iteration of the common model is achieved.

Last, we describe DeepChain corresponding to three proposed definitions in the Blockchain setting, which demonstrates the significance made by DeepChain. $V(.)$ predicate means validation where each $\$Coin$ a party spends in a transaction is guaranteed valid, and a transaction related to DeepChain business is effectively validated. Another predicate $I(.)$ represents how the valid contents of a block are input where workers do effective contribution on processing transactions into blocks. The last predicate $R(.)$ refers to the interpretation of DeepChain where DeepChain records all procedures of each collaborative trained model which is significantly useful for solving current and future AI tasks in a trusted way.

TABLE 1
Notations and implications

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

## 5.2 Concrete Design

Our concrete design includes five building blocks: DeepChain bootstrapping, incentive mechanism, party assert display, cooperative training and consensus protocol. For giving a better understanding, we list related cryptographic notations on TABLE 1.

### 5.2.1 DeepChain bootstrapping

DeepChain bootstrapping accomplishes two things: (1) DeepCoin distribution and (2) genesis block generation which are innate elements for running DeepChain. On one hand, we assume that users have registered on DeepChain and each user uses one of $pk$s as the address corresponding to a DeepCoin unit he possesses or launching a transaction. DeepCoin distribution realizes that each user on DeepChain is allocated amounts of DeepCoins. We assume that the quantity of the allocations are equal.On the other hand, the genesis block contains initial transactions referring to DeepCoin possession statements after the step of DeepCoin distribution. Assume that the round begins with 0 and generates the genesis block. When the genesis block is generated, a random seed $seed_0$ also is public. $seed_0$ is randomly chosen by initial users via using distributed random number generation. We note that $seed_0$ is the base random seed for DeepChain. Particularly, $seed_0$ is one of components to choose the random seed $seed_1$ in the round with the index 1 and the rest of rounds can be done in the same manner. The seeds are crucial to guarantee the randomness to select a new leader who creates a new block. This follows the idea of cryptographic sortition from Algorand [39], [40]. We will introduce it in section 5.2.5 for the health running of DeepChain.

### 5.2.2 Incentive mechanism

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

honestly process parties' transactions where they update parameters of collaborative models with traded gradients.

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

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

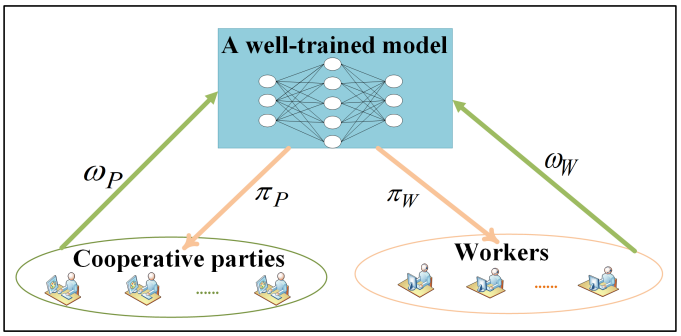


Fig. 3. The incentive mechanism of DeepChain

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

**Payoff=**

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

Second, based on the aforementioned description, we show each party and worker are value-driven to behave honestly in each iteration so that they can obtain the highest payoff, in which the theory derives from the work [41]. We formalize it as $Value(1) = \pi_P - \omega_P(1)$ for a party. Assume that the method is correct with the probability $Pr_v(P)$ to verify a party's malicious behavior is malicious and $Pr_v(W)$ is similar to a worker; the mechanism to launch a penalty is assumed to be designed securely. Note that a party's contribution is defined as $\omega_P$. For party, we say that $\omega_P$ is honestly provided with the probability $Pr_c(P)$. Then, $\omega_P(Pr_c(P))$ is used to represent his true performance. In addition, we get the probability $Pr_v(P) * (1 - Pr_c(P))$ that a dishonest party would be caught. Once the dishonest party is caught, he is punished by forfeiting his deposit, the loss of which is defined as $f_P$. Thus, the returned value according to the party's true behavior can be represented as

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

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

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

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

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

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

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

#### 5.2.3 User assert display

We call data possessed by a user as his assert, and its value is recognized by DeepChain, which is the essential for creating collaborative value. A user who uses DeepChain to find cooperators and accomplish his AI tasks, needs to display his assert first. The user displays assert while protecting data privacy that claims which AI tasks his data is related to. Formally, the user displays his assert via a record which is represented as a transaction including four parts. Note that a transaction is launched by a pseudo public key address. The pseudo public key address is generated by the user according to his wishes. The process to generate pseudo public keys is shown as follows.

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

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

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

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

#### 5.2.4 Collaborative training

Due to the phase of user assert display, parties who have similar AI tasks can constitute a group for collaborative training. In this phase, we introduce the life-cycle of a collaborative training group with four steps, (1) collaborative group establishment; (2) collaborative information commitment; (3) gradient collecting via *Trading Contract*; (4) parameter updating via *Processing Contract*.

• **Collaborative group establishment.** Note that keywords related to AI tasks are displayed in the prior phase. Parties establish a collaborative group $Group$ according to similar "Keywords". They may get more information about "Keywords" by off-line interactions and further confirm their collaboration, the details of which are omitted in the paper. In this step, pre-cooperative parties can audit cooperators' data to ensure the authenticity of data possession. The auditing process is executed based on their displayed data which includes the data proof. Concretely, we present this auditing process in *DataVerifying Contract*. Assume that a party $P_k$ ($pk_{P_k}^{psu}$) wants to audit $P_1$'s data. First, the party selects a subset from $l$, i.e., $set \subset [1, l]$, and requests $P_1$ to give a proof of $\{j, v_j\}_{|set|}$ where $v_j$ is randomly selected from $Z_q^*$ and $j$ is $\in set$. Then, $P_1$ needs to use $v_j$ and compute the proof $(\alpha, \beta)$ for required $\sigma_j$ where $j \in set$. After gaining the proof, $P_k$ can accomplish the auditing by verifying whether the equation $e(\alpha, \beta) = e(\Pi_{j \in set}(H_2(j)^{v_j}) \cdot g^{\beta}, pk_{data\_P_1})$ holds or not. With the accomplishment of security auditing, $N$ parties $P_1, P_2, ..., P_N$ constitute $Group$ with pseudonymity, i.e, pseudo public keys $pk_{P_1}^{psu}, pk_{P_2}^{psu}, ..., pk_{P_N}^{psu}$, and their secret keys $sk_{P_1}, sk_{P_2}, ..., sk_{P_N}$ are privately possessed, respectively. Since parties may launch transactions by using different pseudo public keys, the transactions signed by the same secret key $sk_{P_i}$ can be verified that those transactions are from the same cooperative member.

**Algorithm 1: DataVerifying**($pk_{P_k}^{psu}$, $\{j, v_j\}_{|set|}$)

**receive**($pk_{P_k}^{psu}$, $\{j, v_j\}_{|set|}$) $\quad \#j \in set$
$checkTimeout(T_{a1})$
**getInfoFromDeepChain**() $\rightarrow$
$\{pk_{data\_P_1} = g^{H_1(data\_P_1)}, \sigma_{j\_P_1} = (H_2(j) \cdot g^{H_1(data_{j\_P_1})})^{H_1(data\_P_1)}\}_{|set|}$
**computeProof**() $\rightarrow$
$\{\alpha = \Pi \sigma_{j\_P_1}^{v_j}, \beta = \Sigma v_j \cdot H_1(data_j)\}_{|set|}$
$checkTimeout(T_{a2})$
**verifyProof**($pk_{data\_P_1}, \alpha, \beta$)$\rightarrow Yes \quad or \quad No$
$checkTimeout(T_{a3})$

• **Collaborative information commitment.** After $Group$ establishment, parties agree on their collaborative information for securely training a common deep learning model as follows. In this step, we assume a trusted component only for the setup phase in Threshold Paillier algorithm, and it does not take part in other processes. If there exists no such a trusted component, we can accomplish the setup phase by a distributed way [42].

(1) The number of cooperative parties, $N$.

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

(3) Setup parameters of Threshold Paillier algorithm: $PK_{model} = (n_{model}, g_{model}, \theta = as, V = (v, \{v_i\}_{i \in [1,...,N]}))$ where $g_{model} \in Z_{n_{model}^2}^*$ and

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

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

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

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

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

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

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

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

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

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

After the aforementioned steps, we next introduce how collaborative training is securely accomplished via two smart contracts, *Trading Contract* and *Processing Contract*. Note that collaborative training contains two main functionalities, i.e., iteratively gradient collecting and parameters updating. In DeepChain, traders (parties) iteratively trade their gradients to *Trading Contract* executed by the manager. The trading gradients are honestly encrypted by each trader and meanwhile the correct proofs of encryption are attached, which indicates two security requirements (confidentiality and auditability). In terms of confidentiality, only if a trader does not disclose his gradients, no one can gain any information. Traders (at most $t$ parties) in addition need to cooperatively decrypt parameters after they are updated in the contract. We assume the manager cannot disclose what he knows, as the work [31] promised. In terms of auditability, each trader sending his encrypted gradients also needs to give a correct proof for it. When cooperatively decrypting, each trader also presents his decryption proof. Those proofs are non-interactively public on DeepChain and auditable for any party. On the other hand, the behaviors of traders and the manager are forced to be authentic and fair by utilizing the timeout-checking and monetary penalty mechanisms. Even if the manager colludes with traders, the outcome of *Trading Contract* cannot be modified [31]. On the other hand, *Processing Contract* is responsible for parameter updating. Workers process transactions by casting up gradients in respect to a group, and send computation results to *Processing Contract*. *Processing Contract* verifies correct computation results and updates model parameters for this group. For accomplishing the whole training, these two contracts are called for multiple times. Concretely, we give more detail descriptions for these two contracts in the following steps.

• **Gradient collecting via Trading Contract.** As shown

**Algorithm 2: Trading**($Tran^i_{P_1}$,...,$Tran^i_{P_N}$) #being called by iterations

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

in **Algorithm 2**, it defines six major functions. With those functions being invoked iteratively, gradient transactions for training $model_{com}$ are securely collected and processed. For

the purpose of time-out checking, there declares time points $T_{t_1}, T_{t_2}, T_{t_3}, T_{t_4}, T_{t_5}, T_{t_6}$ following functions, respectively. We stress that the intervals between the time points $T_{t_1}$ and $T_{t_6}$ are declared according to the interval from the time of one iteration $i$ to the time of its next iteration $i+1$, i.e., $|T_{t_6} - T_{t_1}| \leq |T_{i+1} - T_i|$. The time points are set meeting $T_{t_1} < T_{t_2} < T_{t_3} < T_{t_4} < T_{t_5} < T_{t_6}$. At a defined time point, $checkTimeout$ is responsible for checking whether each party behaves honestly or not before the defined time point. If not, the monetary penalty mechanism performs by forfeiting deposits of the malicious parties, and the failed step is re-executed. With being iteratively invoked, time points are updated, e.g., $T^{'}_{t_1} = T_{t_1} + |T_{i+1} - T_i|$.

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

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

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

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

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

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

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

According this, workers make transactions including the newly updated parameters and send them to *Processing Contract* ahead of $T_{t7}$. In the meantime, a leader is randomly chosen from those workers via the consensus protocol on DeepChain. Note that this time the reward for the leader is frozen before verifying his computation work. With $verifyTX$, the correctness of the leader's work is verified by the method that the minority is subordinate to the majority. That is, the result of $C(\mathbf{W}_i)$ given by the leader will be compared with the ones of other competitive workers, and the result is regarded to be correct if the values of the majority are equal to it. If it is incorrect, the leader would be punished according the monetary penalty mechanism, which reduces his reputation on DeepChain, and he gains no reward. Moreover, adapted to DeepChain's consensus protocol (introduced by section 5.2.5), the behavior history influences the probability for him to be a leader on DeepChain. In this case, a leader is re-chosen with his correct computation which is able to be check ahead of $T_{t8}$. At the end, the leader's block collecting correct transactions with correctly updated parameters is appended to DeepChain. For training, the next iteration $(i+1)_{th}$ begins and $model_{i+1}$ is generated based on $model_i$.

**Algorithm 3: Processing()**

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

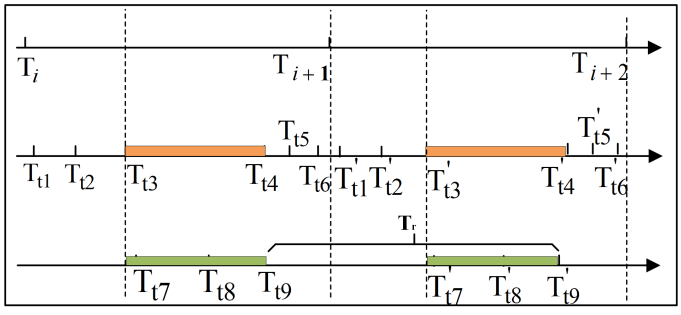


Fig. 4. Configurations on time points. From top to bottom: the timeline of the iterative training, the timeline of trading (in Trading Contract), the timeline of block creation (in Processing Contract).

As shown in *Trading Contract* and *Processing Contract*, the importance of the trusted time lock mechanism is presented. We go back to stress that in *Processing Contract*, time points $T_{t7}, T_{t8}, T_{t9}$ will be updated by $T^{'}_{t7} =$

**Algorithm 4:** $F_{ct}^{*}$ where $ct$ means collaborative training.

**Gradient Collecting**

- Wait to receive a message (input, $sid, T_t, pk_{P_j}^{psu}, C(\triangle \mathbf{W}), Proof_{PK_j}, d(\$Coin)$) from $pk_{P_j}^{psu}$ for all $j \in \{1, ..., N\}$. Assert time $T_t < T_{t_1}$. Here, $sid$ means session identifier and $d(\$Coin)$ means amount of deposits. Then, wait to receive a message (input, $sid, T_t, pk_{P_j}^{psu} \in \mathbb{C}, C(\triangle \mathbf{W}), Proof_{PK_j}$, $H'$, $h' \times d(\$Coin)$) from $\mathcal{S}$ (adversary). Assert time $T_t < T_{t_1}$. Here, $H'$ means the set of the remaining honest parties and $|H'| = h'$.
- Compute $fsver_1(C(\triangle \mathbf{W}), Proof_{PK_j})$ for all $pk_{P_j}^{psu}$ where $j \in \{1, ..., N\}$.
- Record the correct parties as $\{1, ..., N\} \setminus \mathbb{C}'$ according to the computation results.
- Send(return, $d(\$Coin)$) to $pk_{P_j}^{psu}$ for all $j \in \{1, ..., N\} \setminus \mathbb{C}'$ after $T_{t_1}$;
- If $\mathcal{S}$ returns (continue, $H''$) where $H''$ means $H' \setminus \mathbb{C}'$, then send (output, $Yes$ or $No$) to all $pk_{P_j}^{psu}$ where $j \in \{1, ..., N\}$, and send (payback, $(h - h'')d(\$Coin)$) to $S$ where $|H''| = h''$, and send (extrapay, $d(\$Coin)$) to $pk_{P_j}^{psu}$ where $j \in H''$.
- Else if $S$ returns (abort), send (penalty, $d(\$Coin)$) to $pk_{P_j}^{psu}$ for all $j \in \{1, ..., N\}$.

**Collaborative decryption**

-Wait to receive a message (input, $sid, T_t, pk_{P_j}^{psu}, C, C_j, Proof_{CD_j}, d(\$Coin)$) from $pk_{P_j}^{psu}$ for all $j \in \{1, ..., N\}$. Assert time $T_t < T_{t_5}$. Then, wait to receive a message (input, $sid, T_t, pk_{P_j}^{psu} \in \mathbb{C}$, $C, C_j, Proof_{CD_j}$, $H'$, $h' * d(\$Coin)$) from $\mathcal{S}$ (adversary). Assert time $T_t < T_{t_5}$.
- Compute $fsver_2(C, C_j, Proof_{CD_j})$ for all $pk_{P_j}^{psu}$ where $j \in \{1, ..., N\}$.
- Record the correct parties as $\{1, ..., N\} \setminus \mathbb{C}'$ according to the computation result.
- Send(return, $d(\$Coin)$) to $pk_{P_j}^{psu}$ for all $j \in \{1, ..., N\} \setminus \mathbb{C}'$ after $T_{t_5}$;
- If $\mathcal{S}$ returns (continue, $H''$) where $H''$ means $H' \setminus \mathbb{C}'$, then send (output, $Yes$ or $No$) to all $pk_{P_j}^{psu}$ where $j \in \{1, ..., N\}$, and send (payback, $(h - h'')d(\$Coin)$) to $S$ where $|H''| = h''$, and send (extrapay, $d(\$Coin)$) to $pk_{P_j}^{psu}$ where $j \in H''$.
- Else if $S$ returns (abort), send (penalty, $d(\$Coin)$) to $pk_{P_j}^{psu}$ for all $j \in \{1, ..., N\}$.

$T_{t7} + T_r, T_{t8}^{'} = T_{t8} + T_r, T_{t9}^{'} = T_{t9} + T_r$, respectively. $T_r$ means the interval creating a new block within two rounds on DeepChain, as shown in Fig. 4 which depicts the time point configurations involving two contracts. Suppose during $i_{th}$ iteration, defined time points are configured meeting $T_{t_1} < T_{t_2} < T_{t_3} <= T_{t_7} < T_{t_8} < T_{t_9} <= T_{t_4} < T_{t_5} < T_{t_6}$. Meanwhile, the relationship among these three time-lines also shows $Tr \leq |T_{t_6} - T_{t_1}| \leq |T_{i+1} - T_i|$.

In addition to the trusted time lock mechanism, we employ the secure monetary penalty mechanism to present fairness for the procedures of gradient collecting and collaborative decryption. we introduce the formalized methodology proposed by Bentov *et. al* and Kumaresan *et. al* [43], [44] into the **Algorithm 4** $F_{ct}^{*}$ based on *Trading Contract*. In particular, in the process of **Gradient collecting**, fairness is guaranteed twofold: 1) honest collaborative parties must launch gradient transactions which are verified to be correct ahead of the defined time; 2) dishonest parties who launch incorrect transactions or time-out transactions are to be penalized and meanwhile the remained honest are to be compensated. Meanwhile, fairness for collaborative decryption are provided in the process of **Collaborative decryption**, which depicts twofold: 1) a party who gives a correct decryption share at a defined time point never has to pay any penalty; 2) If the adversary successfully decrypt, but a party cannot, then the party should be compensated.

#### 5.2.5 Consensus protocol

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

Particularly, we build the consensus protocol on DeepChain stemmed from Algorand [39], [40] which is a promising blockwise-BA protocol. There exist three main steps: (1) To randomly select a leader who creates a new block by calling cryptographic sortition. (2) A committee verifies and agrees on the new block by executing a Byzantine agreement protocol. The committee is constituted by transaction participants whose transactions are included inside the new block. (3) Each verifier in (2) step tells neighbors the new block via gossip protocol so that the new block is known by all participants on DeepChain. The consensus protocol on DeepChain demonstrates three properties including safety, correctness and liveness, which guarantees the health of DeepChain. Specifically, safety means that all honest parties agree on the same transaction history on DeepChain. Correctness represents that any transaction agreed by any honest party comes from a honest party. Last, liveness says that parties and workers are willing to continuously act on DeepChain, thereby keeping DeepChain living. In order to achieve these three properties, we assume that DeepChain enables synchronous message transmission. With the synchronous network assumption, all parties agree on a chain with the most asserts. With the implementation of step (1), we also assume at most available 2/3 $\$Coin$ are possessed by honest parties. As following we demonstrate the aforementioned three steps, which also demonstrate how the functionalities $V(\cdot), I(\cdot)$, and $R(\cdot)$ (introduced in Section 5.1) work. We define that $r_i$ is referred to the round which creates the block $block_i$.

• **Leader selection** A leader selection means that a block selection. At the round $r_i$, a leader $leader_i$ is randomly

chosen from workers who create the block $block_i$. Before $r_i$ beginning, which $leader_i$ being chosen is unpredictable and random, and after $r_i$ ending, $leader_i$ is public. We call the sortition method of Algorand as form

$Algorand.Sortition(sk, seed_i, \tau = 1, role = worker, w, W) \longrightarrow < hash, \pi, j >$

and

$Algorand.VerifySort(pk, hash, \pi, seed_i, \tau, role = worker, w, W) \longrightarrow j$

for leader selection and leader verification, respectively. Specifically, a pair of $sk$ and $pk$ is owning to a participant. $seed_i$ is random selected based on $seed_{i-1}$ that is $seed_i = H(seed_{i-1}||r_i)$. $\tau = 1$ means we select only one leader from workers $role = worker$. $w$ means amounts of $\$Coins$ with the available validity value the participant possess and those $\$Coins$ without validity value do not be considered, which are different from Alogrand's. $W$ presents the total amounts of $\$Coins$ on DeepChain. Thus, we can randomly select a leader and all participants also enable to verify the selected leader $leader_i$. We set the property of validity value is to limit the trend that is wealth accumulation. It may happen that participants become more rich by accumulating more money due to the higher probability being a leader.

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

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

## 6 Security Analysis

In this section, we recall our security goals in DeepChain which are presented in section 4. We further give our security analyses threefold with respect to three security goals.

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

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

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

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

• **Public auditability for gradient collecting and parameter updating.** The security goal is that any party can audit the correctness of encrypted gradients and decryption shares during the processes of gradient collecting and parameter updating, respectively. Recall that we introduce the non-interactive zero-knowledge proof for these two processes in the setting of the Threshold Paillier algorithm, such as $fsprove_1$, $fsver_1$, $fsprove_2$, $fsver_2$, the methodology of which can be referred to the universally verifiable CDN (UVCDN) protocol [45]. Under the defined framework of UVCDN protocol, public auditability can be guaranteed if there exists a simulator that simulates the correctness proofs of honest parties and extracts witnesses of corrupted parties. We following demonstrate this statement with respect to the correctness proof of encrypted gradients by using ***Lemma*** 2 and ***Lemma*** 3. Note that the proof instances are depicted in **Algorithm** 5 and **Algorithm** 6. Similarly, the correctness proof of decryption shares can be discussed under the UVCDN framework, and we omit this part due to the space limitation.

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

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

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

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

***Lemma 2.*** Given $X = C(x)$, $x = \triangle \mathbf{W}$, and $c \in \mathbb{C}$ where $\mathbb{C}$ is a finite set named the challenge space, Then,
$\{d \in_R Z_{n_{model}}; e \in_R Z^*_{n_{model}}; a := g_{model}^{d} e^{n_{model}} X^{-c}: (a; c; d, e)\}$
$\approx$
$\{a_1 \in_R Z_{n_{model}}; b_1 \in_R Z^*_{n_{model}}; a := g_{model}^{a_1} b_1^{n_{model}}; t := (a_1 + cx)/n_{model}; d := a_1 + cx; e := b_1 k_j^c g_{model}^{t}: (a; c; d, e)\}$

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

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

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

• **Fairness enhancement for collaborative training.** Recall that we employ two security mechanisms in the setting of Blockchain to enhance fairness for collaborative training. The two security mechanisms are trusted time clock and secure monetary penalty mechanisms. Based on the exact timestamp attached to each block which is decentralizedly maintained, to assume the trusted time clock mechanism makes sense. With the mechanism, behaviors in a contract are pushed to be accomplish ahead of a defined time point, which is demonstrated by the function $checkTimeout$ in the setting of DeepChain. On the other hand, we define two secure monetary penalty mechanisms we need, from which one is for gradient collecting and another one is for collaborative decryption. To explain these two mechanisms, we introduce a notion, secure computation with coins (SCC security) in a multi-party $N$ setting, which is defined and proven by [43], [44] in a hybrid model as following.

***Lemma 4.*** Defined input $z$, security parameter $\lambda$, a distinguisher $Z$, ideal process $\texttt{IDEAL}$, ideal adversary $S$ in IDEAL, and ideal function $f$; and meanwhile defined a protocol $\pi$ which interact with ideal function $g$ in a model with adversary $A$, Then,
$\{\texttt{IDEAL}_{f,S,Z}(\lambda, z)\}_\lambda \in \mathbb{N}, z \in 0, 1^*$
$\equiv_c$
$\{\texttt{HYBRID}_{g,\pi,A,Z}(\lambda, z)\}_\lambda \in \mathbb{N}, z \in 0, 1^*$

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

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

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

## 7 Implementation and Evaluation

In this section, we present a implementation prototype for DeepChain, which demonstrates our feasibility. We first build the blockchain setting to simulate DeepChain. With this setting, nodes which are regarded as parties participate in trading, and interact with two defined crucial smart contracts (i.e., *Trading Contract* and *Processing Contract*), in which generated transactions are serialized on the blockchain. Trading parties agree on a common deep learning model at first. This model is encrypted and stored on the blockchain. Then, the parties locally train the model based on individual dataset. They iteratively collect updated gradients into transactions, and send them to the first smart contract. When the training is accomplished, trading interactions also end. During this process, updated gradients in transactions are individually encrypted by each trading parties. With the help of the second smart contract, updated gradients then are used to update the parameters of the common deep learning model. Updated parameters are returned back and collaboratively decrypted by trading parties. Until the model training is accomplished, trading parties iteratively launch transactions as the above processes.

First, we choose Corda project to simulate DeepChain for adaption and simplification. Corda project is created by R3CEV, as well as widely applied in bank, financial institutes and trading areas. It is a decentralized ledger which absorbs the features of Bitcoin and Ethereum while creating its characteristics, such as data sharing based on need-to-know basis, deconflicting transactions with pluggable notaries. A Corda network contains multiple notaries where the consensus protocol introduced in section 5.2.5 can be executed for them. Though we do not implement this in this paper, we make it for our further work. Without the loss of generation, we build nodes and classify them into two kinds, parties and workers. They constitute into the nodes of two CorDapps agreeing on the blockchain, in which we define different business logic in five components, such as Flows, States, Contracts, Services and Serialisation whitelists.

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

TABLE 2
Training configuration

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

$b_2 = (1, 128)$, $w_3 = (128, 10)$ and $b_3 = (1, 10)$, respectively. Additionally, other training parameters are configured as the table 2 shown.

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

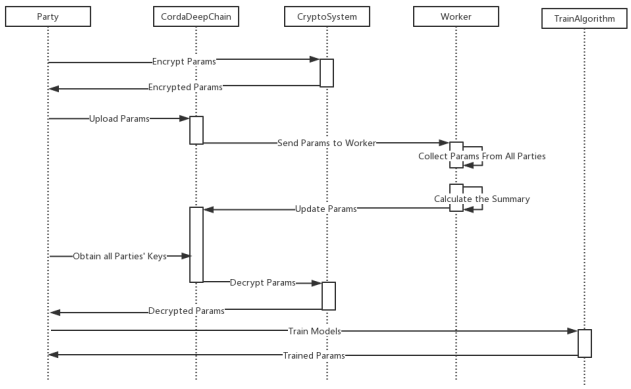


Fig. 5. The interaction activities within the implemented modules.

We implement the aforementioned building blocks with three modules, CordaDeepChain, TrainAlgorithm and CryptoSystem, respectively. For accomplish a training process, their interaction activities are as shown in Fig. 5. We evaluated the feasibility of training on the simulated DeepChain in terms of encryption and training performance in a multi-party setting. First of all, we evaluate encryption performance with the implemented program on a desktop which is an Intel(R) Xeon(R) CPU machine with 3.30 GHz cores and 16 GB memory. Fig. 6 shows the size of cipher is a constant when we encrypt various amounts of gradients which means the number of elements in the vector to be encrypted. Then, Fig. 7 shows the throughput when the encrypt algorithm is executed.

On the other hand, we create four parties participating in collaborative training and trading. Each party trains the local model with the training dataset which has the size of 13750 (by $55000/4$). Then, single party gains the averaged gradients shared from the other three parties. We also create an external party only training on 13750-size dataset without the sharing averaged gradients, which is regarded as a baseline party. Through making the training accuracy comparison between the results from collaborative parties and the base line party, We demonstrate the accuracy improvement for single collaborative parties. The comparison result is shown in Fig 8.

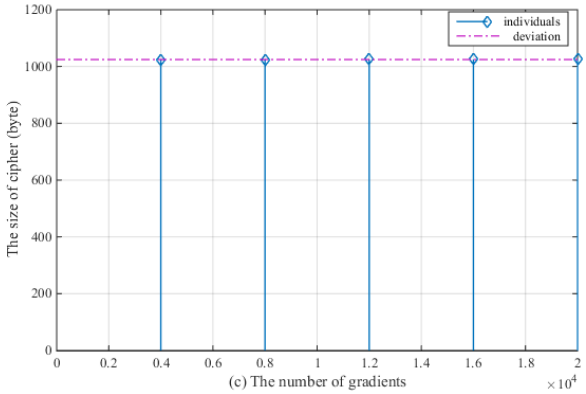


Fig. 6. Evaluation on the cipher size.

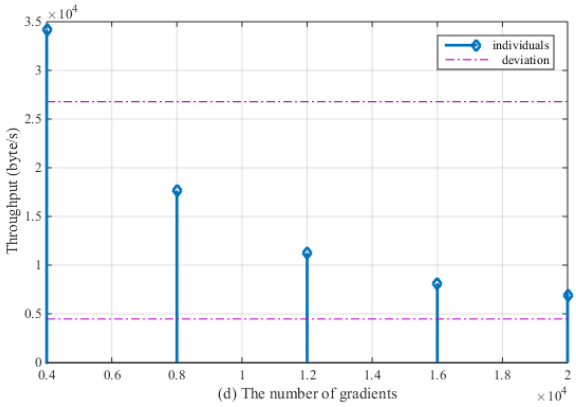


Fig. 7. Evaluation on the encryption throughput.

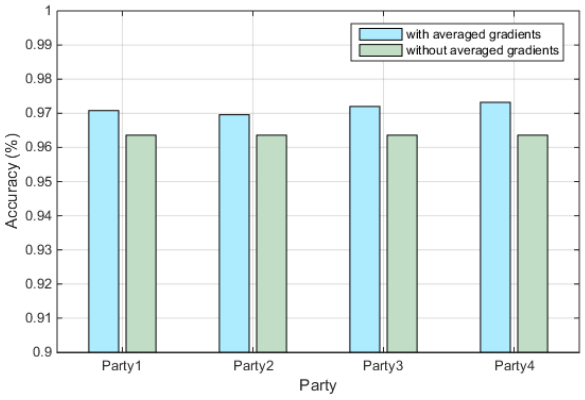


Fig. 8. The comparison on the training accuracy.

## 8 Conclusion and Future Work

In this paper, we present DeepChain, which is a healthy and win-win decentralized platform based on Blockchain for secure deep learning training. In the setting of federal learning, we introduce an incentive mechanism and meanwhile focus three security goals that are confidentiality, auditability as well as fairness. In addition, we claim the value

of DeepChain in a long-term way. DeepChain stores training models where not only iterative training parameters but also trained models are recorded. On the one hand, it is obvious that trained models create financial values when the model-based pricing market is promising. This brings the owners of trained models with long-term benefits, since their models can serve for those who have AI tasks by the way of payment. On the other hand, all training processes and well-trained models are recorded, which could advance the development of transfer learning. Andrew Ng, in NIPS 2016 tutorial has said: "Transfer learning will be the next driver of ML success." [46] Thus, we take the first-step consideration that DeepChain can extend model values to transfer learning. Trained models which have gained knowledge solving one problem can be applied to a different but related problem. Then, the security problem, such as the privacy issue can be modeled, and in the case of model-to-model this issue could be discussed in the future work.

## ACKNOWLEDGEMENT

This work was supported by National Science Foundation of China (Grant Nos. 61472165 and 61373158), Guangdong Provincial Engineering Technology Research Center on Network Security Detection and Defence (Grant No. 2014B090904067), Guangdong Provincial Special Funds for Applied Technology Research and development and Transformation of Important Scientific and Technological Achieve (Grant No. 2016B010124009), the Zhuhai Top Discipline–Information Security.

## REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[17] D. Das, S. Avancha, D. Mudigere, K. Vaidynathan, S. Sridharan, D. Kalamkar, B. Kaul, and P. Dubey, "Distributed deep learning using synchronous stochastic gradient descent," 2016.

[18] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," 2016.

[19] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, and P. Tucker, "Large scale distributed deep networks," in International Conference on Neural Information Processing Systems, 2012, pp. 1223–1231.

[20] N. Feng, B. Recht, C. Re, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," Advances in Neural Information Processing Systems, vol. 24, pp. 693–701, 2011.

[21] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, "Parallelized stochastic gradient descent," in Advances in Neural Information Processing Systems 23: Conference on Neural Information Processing Systems 2010. Proceedings of A Meeting Held 6-9 December 2010, Vancouver, British Columbia, Canada, 2010, pp. 2595–2603.

[22] B. C. Ooi, K. L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, Y. Wang, and Y. Wang, "Singa: A distributed deep learning platform," in ACM International Conference on Multimedia, 2015, pp. 685–688.

[23] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015, pp. 1355–1364.

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

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

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

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

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

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

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

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

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

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

[34] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," arXiv preprint arXiv:1501.02876, vol. 7, no. 8, 2015.
[35] M. Lin, S. Li, X. Luo, and S. Yan, "Purine: A bi-graph based deep learning framework," arXiv preprint arXiv:1412.6249, 2014.
[36] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Inference attacks against collaborative learning," arXiv preprint arXiv:1805.04049, 2018.
[37] L. Chen, P. Koutris, and A. Kumar, "Model-based pricing for machine learning in a data marketplace," arXiv preprint arXiv:1805.11450, 2018.
[38] A. B. Kurtulmus and K. Daniel, "Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain," arXiv preprint arXiv:1802.10185, 2018.
[39] S. Micali, "Algorand: The efficient and democratic ledger," arXiv preprint arXiv:1607.01341, 2016.
[40] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017, pp. 51–68.
[41] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya, "Incentivizing outsourced computation," in Proceedings of the 3rd international workshop on Economics of networked systems. ACM, 2008, pp. 85–90.
[42] T. Nishide and K. Sakurai, "Distributed paillier cryptosystem without trusted dealer," in International Workshop on Information Security Applications. Springer, 2010, pp. 44–60.
[43] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in International Cryptology Conference. Springer, 2014, pp. 421–439.
[44] R. Kumaresan and I. Bentov, "How to use bitcoin to incentivize correct computations," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2014, pp. 30–41.
[45] B. Schoenmakers and M. Veeningen, "Universally verifiable multiparty computation from threshold homomorphic cryptosystems," in International Conference on Applied Cryptography and Network Security. Springer, 2015, pp. 3–22.
[46] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," arXiv preprint arXiv:1701.00160, 2016.