future internet

Article

TDLearning: Trusted Distributed Collaborative Learning Based on Blockchain Smart Contracts

Jing Liu [1,*], Xuesong Hai [1] and Keqin Li [2,*]

[1] College of Computer Science, Inner Mongolia University, Hohhot 010021, China; 32009008@mail.imu.edu.cn
[2] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA
* Correspondence: liujing@imu.edu.cn (J.L.); lik@newpaltz.edu (K.L.)

Abstract: Massive amounts of data drive the performance of deep learning models, but in practice, data resources are often highly dispersed and bound by data privacy and security concerns, making it difficult for multiple data sources to share their local data directly. Data resources are difficult to aggregate effectively, resulting in a lack of support for model training. How to collaborate between data sources in order to aggregate the value of data resources is therefore an important research question. However, existing distributed-collaborative-learning architectures still face serious challenges in collaborating between nodes that lack mutual trust, with security and trust issues seriously affecting the confidence and willingness of data sources to participate in collaboration. Blockchain technology provides trusted distributed storage and computing, and combining it with collaboration between data sources to build trusted distributed-collaborative-learning architectures is an extremely valuable research direction for application. We propose a trusted distributed-collaborative-learning mechanism based on blockchain smart contracts. Firstly, the mechanism uses blockchain smart contracts to define and encapsulate collaborative behaviours, relationships and norms between distributed collaborative nodes. Secondly, we propose a model-fusion method based on feature fusion, which replaces the direct sharing of local data resources with distributed-model collaborative training and organises distributed data resources for distributed collaboration to improve model performance. Finally, in order to verify the trustworthiness and usability of the proposed mechanism, on the one hand, we implement formal modelling and verification of the smart contract by using Coloured Petri Net and prove that the mechanism satisfies the expected trustworthiness properties by verifying the formal model of the smart contract associated with the mechanism. On the other hand, the model-fusion method based on feature fusion is evaluated in different datasets and collaboration scenarios, while a typical collaborative-learning case is implemented for a comprehensive analysis and validation of the mechanism. The experimental results show that the proposed mechanism can provide a trusted and fair collaboration infrastructure for distributed-collaboration nodes that lack mutual trust and organise decentralised data resources for collaborative model training to develop effective global models.

Keywords: distributed collaborative learning; smart contract; model fusion; formal verification

check for updates

Citation: Liu, J.; Hai, X.; Li, K. TDLearning: Trusted Distributed Collaborative Learning Based on Blockchain Smart Contracts. *Future Internet* **2024**, *16*, 6. https://doi.org/10.3390/fi16010006

Academic Editor: Ashutosh Dhar Dwivedi

Received: 22 November 2023
Revised: 17 December 2023
Accepted: 21 December 2023
Published: 25 December 2023

1. Introduction

Data are important catalysts for the development of Artificial Intelligence (AI), and the quantity and quality of data determines the upper limit of the performance of AI models [1]. The combination of blockchain and federated learning is a highly promising research field, especially in addressing issues related to data privacy, security and trustworthiness [2,3]. Yang et al. [4] proposed an explainable federated learning architecture and a blockchain-based credit-scoring system (EFCS). This method uses an improved federated learning mechanism that enhances its explainability while increasing the reliability of training results and employs credit-qualification proof and credit-evaluation proof to further verify the correctness of the training results. Our research differs from that of Yang et al. in that we focus more on the definition of collaborative behaviours, relationships and norms among distributed cooperative nodes through blockchain smart contracts. Based on this definition and transfer learning, we propose a feature-fusion-based model-fusion method. Similarly to their approach, we replace the direct sharing of local data resources with distributed-model collaboration training to organise scattered data resources for distributed cooperation, thereby improving model performance. However, our work does not involve an improvement in federated learning, and smart contracts play a more significant role in the collaboration process. However, in practical application scenarios, data resources are often highly dispersed. In addition, due to data privacy and security constraints, it is difficult for multiple data sources to share their local data directly. It is difficult to effectively aggregate data resources, which will lead to a lack of support for model training. This problem restricts the further improvement in the model performance. To address these challenges, distributed-collaborative-learning architectures based on dispersed data have emerged. For example, the federated learning (FL) proposed by Google [5], which implements an efficient distributed-collaboration model based on decentralised data and computing power. Although existing distributed-collaborative-learning architectures have made a lot of efforts in privacy protection [6], incentive mechanisms [7] and protection against malicious attacks [8], they still assume that the collaboration is mainly performed between trusted environments and mutually trusted collaboration nodes and rely to some extent on third-party platforms. Security and mutual distrust have seriously affected the confidence and willingness of data owners to collaborate. Therefore, how to ensure the credibility and fairness of the collaboration process has become the core issue of distributed collaboration based on decentralised data.

The development of blockchain technology [9] has filled a gap in many fields and also brought development opportunities for distributed-collaborative-learning architecture. As a decentralised infrastructure, blockchain provides trusted distributed storage with good features such as the immutability, traceability and security of stored content. The application of smart contracts [10] has greatly enhanced the scalability of the blockchain platform, giving it Turing-complete programming capabilities and providing trusted distributed computing.

Locating the problem of scattered data resources and the low trustworthiness of existing distributed-collaborative-learning architectures, we propose a trusted distributed-collaborative-learning mechanism based on blockchain smart contracts. The mechanism uses the blockchain as a distributed collaborative execution environment, and the cooperative behaviour and specifications are defined and encapsulated by smart contracts. At the same time, a model-fusion method based on feature fusion is proposed to realise model collaborative training to improve the model performance. The mechanism strives to provide a trusted and fair distributed-collaboration infrastructure, enabling all collaboration parties to perform trusted distributed collaboration in the absence of mutual trust and third-party intermediary dependence.

The innovations and contributions of this paper are as follows:

- We use smart contracts to define and encapsulate collaborative behaviours, relationships and specifications between distributed-collaboration nodes.
- Based on the idea of smart contract collaboration architecture and transfer learning, we propose a model-fusion approach based on feature fusion to replace the direct sharing of local data resources with distributed-model collaboration training.

The remainder of this paper is organised as follows. Section 2 summarises the work involved. Section 3 details the architecture and design details of the collaborative-learning mechanism. In Section 4, we implement a formal verification of the mechanism to prove its trustworthiness. In Section 5, the usability of the mechanism is demonstrated by means of experiments and a case study. The conclusion is presented in Section 6.

2. Related Work

At present, many researchers are committed to combining blockchain technology with distributed-collaborative-learning architecture. Flexible consensus protocols, effective incentives, privacy protection, data and model management, smart contract applications and the scalability of distributed architectures are the key directions of current research [2,11–13].

2.1. Replacing the Central Server in Distributed Learning

Traditional distributed-collaborative-learning architectures rely on a single central parameter aggregation server, so the architecture faces single points of failure and server trust issues. Kim et al. [14] proposed using blockchain instead of parameter servers in federated learning. It treats each collaboration client as a blockchain node, and the client uploads parameters to the blockchain after completing local training. The miner records and completes the model-parameter verification, generates a new block through the proof-of-work (POW) consensus mechanism and then each client obtains the parameter update from the blockchain and completes the global model update locally. Qu et al. [15] made improvements in privacy protection, the client's local-model parameters are encrypted and uploaded to the blockchain and the global model parameters need to be codecrypted by the collaborative client from the block to prevent information leakage. While using differential privacy techniques to protect data privacy, Wang et al. [16] designed reinforcement-learning-based incentive mechanisms to improve the usability of blockchain-based federated learning frameworks. Lu et al. [17] noted that applying POW to model-parameter sharing can cause long delays. Therefore, this paper proposes to replace POW with a proof-of-quality (POQ) consensus mechanism to reduce the cost of communication and reduce the impact of low-quality models on collaboration.

The above method uses the blockchain to replace the central server or increase its credibility by integrating the parameter-interaction process into the underlying mining mechanism of the blockchain. However, maintaining blockchain nodes requires a lot of storage resources, and the limited resources of client devices may become a bottleneck for collaboration. In addition, the complexity of this type of method is high, and the modification of the consensus protocol will inevitably reduce the versatility and scalability of the method.

2.2. Recording Distributed-Collaboration Processes

Harris et al. [18] proposed a framework for building datasets by participants, in which data and global models are shared by all collaborators. The framework uses smart contract hosting to continuously update the global model and gives incentives to participants based on the improvement in the model accuracy provided by the data. Lugan et al. [19] proposed using blockchain to record the iterative version of the global model in distributed learning, and when the global model degraded, it could be traced back in time. Awan et al. [20] recorded all interactions in federated learning in the blockchain, such as FL tasks, local-model sources and global model update information. The blockchain has the traceability and tamper-proof characteristics of stored content, which enables the central server to safely evaluate the contribution of each collaboration client to the global model update and motivate relevant collaboration clients. Miao et al. [21] implemented homomorphic encryption-based federated learning, where the encrypted intermediate parameters are stored in a blockchain in order to prevent the malicious tampering of the parameters. Ma et al. [22] proposed a layered federated learning architecture based on blockchain, which divides FL into a network layer, a blockchain layer and an application layer. The network layer consists of peer-to-peer collaborative clients, where local-model updates and global-parameter aggregation are performed by the collaborative clients. The blockchain layer is responsible for the permanent storage of collaborative information in the form of transactions, and the application layer is defined by smart contracts, which are responsible for the publication of FL tasks and the selection of collaborative clients.

The above literature uses blockchain as a distributed ledger to record distributed-collaboration information, which enhances the security of collaboration and provides credible evidence for incentive allocation and malicious-behaviour recovery. However, this area of research does not typically improve the collaboration process of existing distributed-collaboration architectures, and the recording of collaborative interactions does not fully improve the untrustworthy collaboration environment.

2.3. Distributed-Collaborative-Learning Mechanism Based on Blockchain Smart Contracts

Bozkurt et al. [23] and Alsobeh et al. [24] are concerned about the problems of decentralised collaboration in terms of its practical application and are attempting to address them by using blockchain technology. Ramanan et al. [25] proposed using smart contracts to implement model-parameter aggregation in federated learning and control global model iteration and other steps. However, it may not be practical to completely replace a central parameter server with smart contracts, and existing blockchain platforms such as Ethereum cannot effectively support compute-intensive tasks. Mendis et al. [26] proposed a blockchain-based decentralised computing paradigm and put forward the idea of using blockchain to control distributed collaborative processes but did not address the details of process design and the concrete implementation of smart contracts. In recent years, distributed AI collaboration architectures based on smart contract designs have been proposed, with research centred on the fair and trusted organisation of distributed computing resources and the formation of an auditable marketplace for AI-pretrained models [27,28]. Oktian et al. [29] designed a complete protocol based on federated learning by using smart contracts, including features such as incentives, model validation and collaborator-reputation assessment, to address the problems of low client motivation to collaborate, midway withdrawal and unauthorised model access in federated learning. The protocol is functional; however, on-chain interactions will further increase the communication cost of federated learning, and smart contracts may become a computational bottleneck.

In summary, the design of distributed-collaborative-learning mechanisms based on blockchain smart contracts still has great research potential, and how to efficiently organise decentralised data resources in the collaborative mechanism still needs further research. Therefore, we aim to use blockchain as a trusted collaboration environment; give full play to the unique advantages of smart contracts in realizing identity authentication and authority management, trusted storage, incentive mechanism and data and model management in distributed collaboration; and design a trusted distributed-collaborative-learning mechanism based on decentralised data resources.

3. Method

This section first introduces the overall architecture of TDLearning, followed by a detailed description of the smart contract design and key algorithms, and finally proposes and analyses a model-fusion approach based on feature fusion.

3.1. Architecture of TDLearning

Effective collaboration based on decentralised data resources and the use of smart contracts to ensure a trustworthy and fair collaborative process are the goals of the collaborative-learning mechanism proposed in this paper. Blockchain smart contracts are utilised to establish and capture collaborative behaviours and connections in this paper. Specifically, in this study, the distributed-collaboration process of the collaborative-learning mechanism is defined as a collaborative task, and a smart contract defines and encapsulates the collaborative relationship between the actors and nodes in the task, acting as a medium of communication and trust. At the same time, a model-fusion method based on feature fusion is proposed based on the idea of migration learning. Instead of the direct sharing of data resources, distributed-model collaboration training is used to organise dispersed data resources for effective distributed collaboration.

The overall architecture of TDLearning is shown in Figure 1. The mechanism consists of six smart contracts, which are defined and encapsulated into three types of functional modules: role-authority management, trusted database construction, collaborative task management and collaborative incentive allocation. Authority management is implemented by the authority-management contract (AMC), database construction is based on the data-retrieval contract (DRC) and the data-evaluation contract (DEC) and task-process management and motivation are based on the task-management contract (TMC) and the task-incentive contract (TIC). The node in the mechanism contains four types of roles, namely supervisor, task owner, task participants and model verifiers. The roles and their corresponding collaborative behaviours are shown in Table 1. From the perspective of each node participating in the collaborative training of the distributed model, the collaborative-learning mechanism consists of the following three steps in total.

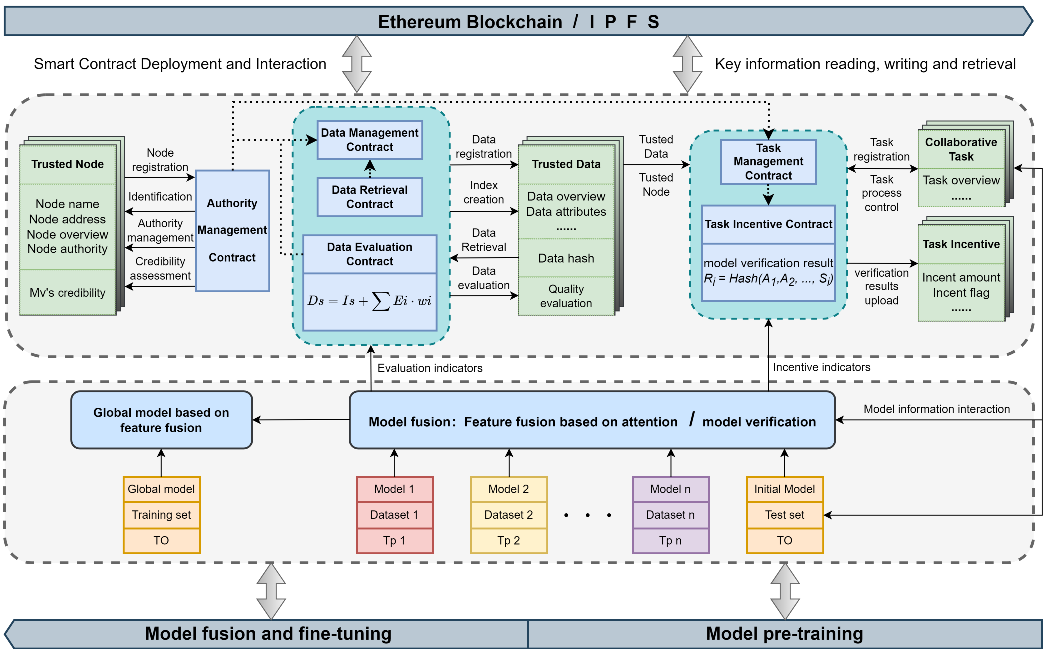


Figure 1. Overall architecture of TDLearning: (1) Smart contracts define and encapsulate collaborative relationships, behaviours and specifications between collaborating nodes. (2) Collaborative model training: a model-fusion approach based on feature fusion.

Table 1. The description of notation symbols used in TDLearning.

Role	**Collaborative Behaviours**
Supervisor—*Su*	(1) Smart contract deployment and collaborative node authority management. (2) Collaboration task access audit and collaboration node on-chain behaviour monitoring. (3) Initiation of smart-contract-function execution, such as data-quality assessment, contract incentive, etc.
Task Owner—*To*	(1) *To* issues global-model-development tasks, specifying pretrained model structures as well as standard test sets. (2) *To* registers on-chain tasks via smart contracts and completes the incentive funding pledge. (3) After receiving all the validated pretrained models, *To* completes global model fusion and fine tuning locally.
Task Participant—*Tp*	(1) *Tp* registers local private data by using a data-management contract, which participates in the collaborative-learning task as a data node. (2) *Tp* uses the specified model structure to develop pretrained models locally by using the private data. (3) After the pretrained model is verified, *Tp* receives automatic motivation from the smart contract.
Model Verifier—*Mv*	(1) *Mv* completes pretrained model verification based on the standard test set given by the task owner and uploads the validation results via the smart contract. (2) After the smart contract reaches a consensus on the pretrained model validation, *Mv* receives an automatic incentive from the smart contract.

1. *Tp* uses its local data to develop pretrained models.
2. *Mv* verifies all pretrained models by using the standard validation set provided by *To*.
3. *To* receives a validated pretrained model, and model fusion is completed by using its local data to develop the global model.

The three types of functional modules defined and encapsulated by smart contracts locate and solve each of the following three types of problems:

- The role-authority-management module locates the trust issue for collaborative-learning nodes, which maintain a list of trusted collaborative nodes through the AMC. In this mechanism, the prerequisite for participation in collaboration is the registration and authentication of identity, and role-based authority management assigns authority to collaborative nodes by role. The binding of identity information to authority encourages collaborative nodes to act honestly, and the traceability and tamper-evident nature of the blockchain strongly restrains them from acting maliciously.
- Trusted database construction locates data-availability and collaborative-data-quality issues. The DRC maintains an index of registered data, which enables the on-chain retrieval of data. To achieve the keyword retrieval of blockchain data, the traditional solution is to synchronise data off-chain for retrieval; however, due to the existence of synchronisation cycles and the fact that off-chain data security cannot be guaranteed, this will inevitably lead to a reduction in the data availability, hence the need for secure on-chain retrieval in this mechanism. The DEC introduces a fuzzy comprehensive evaluation method, which uses the information in the data and the collaborative behaviour of the data nodes as indicators to evaluate the quality of the data. The DMC completes data registration and maintains a list of trusted data to build a trusted database by invoking the DRC and DEC.
- The collaborative task-management and collaboration-incentive-allocation modules locate the credibility and fairness of collaboration. The TMC controls the process of task execution by differentiating the status of collaborative tasks and maintaining task lists. At the same time, to ensure the quality of pretrained models and the fairness of collaboration, we introduce a verification mechanism for the validity of pretrained models, in which the smart contract determines whether multiple model verifiers have reached a consensus on model verification. Model-verification consensus ensures verification fidelity and pretrained model validity and provides incentive weighting metrics for collaborative incentive allocation. Once consensus is reached, the TIC assigns incentives with predetermined rules and model-verification results. In summary, the collaboration mechanism uses smart contracts to implement distributed-collaborative-learning-task interaction control to guarantee collaborative trustworthiness.

In order to achieve distributed collaboration and reduce the risk of data-privacy leakage, the collaborative-learning mechanism transforms the value of data resources into the value of pretrained models and replaces the exchange and sharing of local data among collaborators with the fusion of pretrained models. Therefore, the effective fusion of multiple pretrained models is the key to the effective organisation of distributed data resources in collaborative-learning mechanisms. In view of the above problems and background, inspired by migration-learning ideas, we propose a model-fusion method based on feature fusion to achieve distributed-model collaborative training. As shown in Figure 2, task participants complete submodel pretraining by using their local data, and the task owner generates global features by fusing the features extracted from the pretrained submodels and uses the global features as classifier inputs to develop global models.

3.2. Task Flow of TDLearning

As shown in Figure 3, the collaborative-learning-task process is divided into a task-information-initialisation phase and a task-execution phase from the perspective of each node participating in the collaborative training of the distributed model. The core steps of the task-information-initialisation phase are as follows.

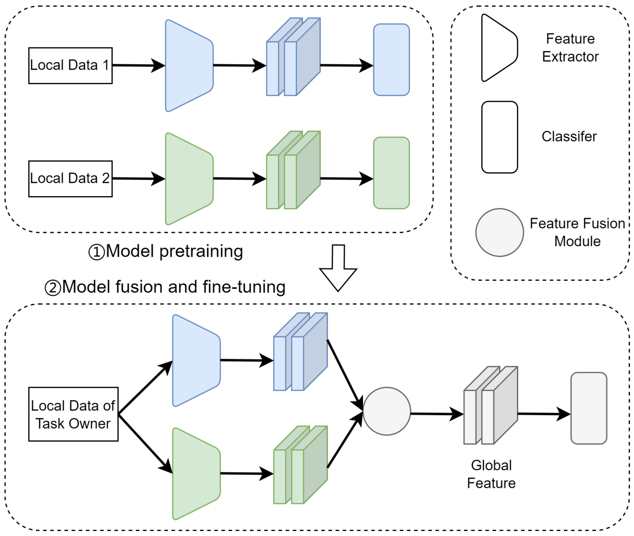


Figure 2. Model-fusion method based on feature fusion.

1. *Su* deploys the relevant smart contracts and assigns a set of *Mv* to initialise their credibility ratings.
2. Each collaborating node registers identity information through the AMC, and the *Su* assigns the nodes corresponding to role authority based on the identity request information.
3. *Tp* registers the local data summary via the DMC. The DMC invokes the DRC and DEC to index the newly registered data and initialise its quality evaluation.
4. *To* registers and initialises collaborative tasks via the TMC, which specifies the task overview, pretrained model structure, standard test sets and *Mv* credibility threshold, and pledges task incentives to the TMC. *To* uses the smart contract to perform secure on-chain searches and queries the quality ratings of relevant datasets to find high-value trusted data.
5. *To* specifies the data nodes involved in the task, i.e., the task participants, while the supervisor specifies the model verifiers for the task based on the *Mv* credibility threshold requirement. After the above steps are completed, if the TMC determines that sufficient collaboration incentives have been deposited, the collaboration task is formally published.

The core steps of the task-execution phase are as follows.

1. Tp_s develop a pretrained model locally by using private data in the specified model structure. Once the pretrained model is developed, Tp_s send it to the specified Mv_s as an on-chain encrypted transmission.
2. Mv_s complete pretrained model verification locally with a standard test set specified by *To* and upload the verification results encrypted to the TMC.
3. The TMC determines whether a consensus on pretrained model verification is reached based on the verification results of Mv_s. When the consensus is reached, Mv_s send the pretrained model to *To* via on-chain encryption. At the same time, the TMC creates the TIC, and the TIC completes the collaboration-incentive allocation, and both Tp_s and Mv_s involved in the collaboration will be automatically incentivised by the smart contract.
4. *To* invokes the DEC to update the collaborative data-quality evaluation. At the same time, *To* invokes the AMC to update the credibility evaluation of Mv_s. If a collaborative node believes that *Mv* has suspicious malicious behaviour, the collaborative node can request to initiate a trust arbitration vote on the *Mv*, and the AMC updates the *Mv* credibility evaluation based on the vote arbitration result.

5. *To* completes pretrained model fusion to develop the global model, and the collaborative task ends.

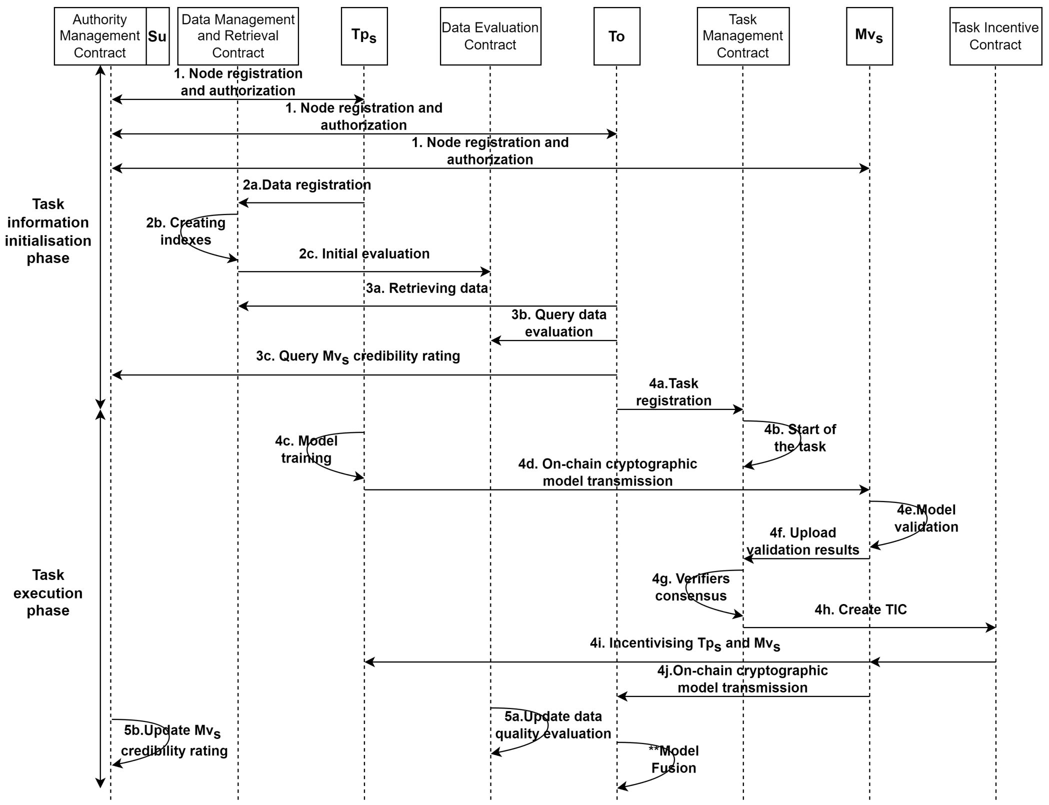

Authority Management Contract
Su
Data Management and Retrieval Contract
Tp_s
Data Evaluation Contract
To
Task Management Contract
Mv_s
Task Incentive Contract
Task information initialisation phase
Task execution phase
1. Node registration and authorization
1. Node registration and authorization
1. Node registration and authorization
2a.Data registration
2b. Creating indexes
2c. Initial evaluation
3a. Retrieving data
3b. Query data evaluation
3c. Query Mv_s credibility rating
4a.Task registration
4b. Start of the task
4c. Model training
4d. On-chain cryptographic model transmission
4e.Model validation
4f. Upload validation results
4g. Verifiers consensus
4h. Create TIC
4i. Incentivising Tp_s and Mv_s
4j.On-chain cryptographic model transmission
5a.Update data quality evaluation
5b.Update Mv_s credibility rating
**Model Fusion

Figure 3. Sequence diagram of trusted distributed cooperative-learning mechanism based on smart contract.

3.3. Smart Contract Design

Smart contracts act as a medium of trust, a guarantee of trustworthiness and fairness for distributed-collaborative-learning mechanisms, and this section details their key algorithmic design. Table 2 shows the key notations used in TDLearning and their descriptions.

Table 2. Notation symbols used in TDLearning.

Notation	Description
$N_{address}$	Collaborative node Ethereum address
N_{info}	Node identity and authority information
N_{role}/N_{status}	Node role/status
$N_{name}/N_{summary}$	Node name/summary
D_{hash}/D_{status}	Data hash/availability status
$D_{name}/D_{summary}$	Data name/summary
T_{name}	Collaborative task name
$T_{summary}/T_{reward}$	Task summary/incentive benchmark
T_{secret}	Cryptographic numbers for model verification
V_n	The number of Mv_s specified by *To*
C_v	Credibility for Mv
V_θ	Credibility threshold for Mv_s
θ	Model-verification thresholds

3.3.1. Role-Authority Management

As shown in Algorithm 1, the role-authority-management module is implemented by the AMC, which implements a role-based authority-management mechanism for accessing and assigning authority to collaborative nodes to participate in collaboration. Its core function is to maintain a list of trusted collaborative nodes. In addition, the Mv is a special collaborative role, which interacts with both the To and the Tp. Therefore, the AMC records the list of Mv_s and maintains credibility values for them. When the To and Tp_s believe that the Mv is behaving suspiciously, they can apply to the Su to initiate arbitration, and the AMC will initiate a trust arbitration vote on the Mv.

Algorithm 1 Authority Management (AMC)

```
1. nodeRegister():
if ($msg.sender$ is not registered)
    Update node list: $N_{info}[msg.sender]$ = nodeInfo ($N_{role}$, $N_{status}$, $N_{name}$, $N_{address}$, $N_{summary}$);
    Initialise node authority;
else
    return false;
2. updateAuthority():
require ($msg.sender_{role} == Su$)
    Update $N_{info}$ [$N_{address}.nodeRole$] = $N_{role}$;
    Update $N_{info}$ [$N_{address}.nodeStatus$] = $N_{status}$;
3. updateVerifiersCredit():
require ($msg.sender_{role} == Su$)
    for each $i$ in $Mv_s$
        if ($i$ is not been applied for arbitration)
            $Cv_i = Cv_i + 1$;
        else
            $Su$ creates the arbitration for $i$;
```

(a) A new node participating in a collaborative task calls nodeRegister() to register identity information with the blockchain. N_{role} is represented by four sets of key–value pairs, 0 for an undefined role, 1 for Su, 2 for To and Tp and 3 for Mv. N_{status} is represented by two sets of key–value pairs, 0 for pending audit or illegal status and 1 for legal status. $N_{address}$ is a unique node identifier, which is bound to the identity-authority information by keyword mapping.

(b) Su calls updateAuthority to edit collaborative node permissions. The blockchain keeps a realistic record of collaborative interactions, allowing Su to intervene in a timely manner when a node behaves suspiciously.

(c) Su deploys smart contracts while initializing Mv and initializing C_v. To deal with potentially malicious behaviour, the collaborative-learning mechanism introduces a model verifier credibility-evaluation system and an arbitration voting mechanism for suspicious behaviour. After the collaborative task is completed, Su calls the updateVerifiersCredit() to update C_v. C_v is updated as shown in Equation (1), where n is the number of untrusted votes for Mv in the arbitration vote and n is 0 to mean that no node in this task has requested to initiate arbitration for Mv. m is the number of Tp_s, and Su removes the Mv from the verifiers list and sets C_v to 0 when more than half of the Tp_s believe that the Mv has suspicious behaviour:

$$C_v = \begin{cases} C_v + 1, n = 0 \\ C_v - n, n \leq \frac{m}{2} \\ 0, n > \frac{m}{2} \end{cases} \tag{1}$$

3.3.2. Trusted Database Construction

Trusted database construction is implemented by the DMC, DRC and DEC. The DRC creates an inverted index for on-chain data to support the keyword retrieval of data. The on-chain data retrieval returns the latest status of the data in real time, enhancing the data availability of the collaboration mechanism, which therefore eliminates the need to maintain an off-chain synchronised database. The data-assessment contract introduces a fuzzy comprehensive evaluation mechanism to evaluate the quality of the data involved in the collaboration. This evaluation process is automatically completed by the smart contract, and the relevant data-quality evaluation is updated at the end of each collaborative task. The DMC implements data registration on the chain. Its core function is to maintain the list of available data and to build a high-value trusted database by calling the DRC and DEC. The core steps are described below:

(a) Data registration. *Tp* calls the dataRegister() to register data information with the blockchain. D_{hash} is a unique identifier of the data in the data list, which is bound to the rest of the data information by keyword mapping.

(b) Data indexing. The blockchain continuously listens for the data-registration event "*event DataReg(D_{hash})*", and the DMC uses this to determine whether the corresponding data have been registered. When a collaborating node completes the registration of new data, the smart contract creates an inverted index for it. The detailed steps for index building are shown in Algorithm 2, which consists of two steps: data summary preprocessing and building the inverted index.

Algorithm 2 Data registration and retrieval (DMC and DRC)

```
1. dataRegister():
if msg.sender has audited and these data are not registered then
   Update database: DataSet[$D_{hash}$] = dataDescription ($D_{name}$, $D_{summary}$, $D_{hash}$, $D_{status}$);
else
   throws;
end if
2. createDataIndex(): when the blockchain has listened for data-registration event:
lexicon[] = processString ($D_{summary}$);
for all each word i in lexicon[] do
   if 0 == bytes (lexicon[i]) or stopWords[lexicon[i]] then
      continue;
   else
      word = stem (lexicon[i]);
      if 0 == invertedIndex[word]. termFrequency ($D_{hash}$) then
         invertedIndex[word]. hashList.push ($D_{hash}$);
         invertedIndex[word]. termFrequency ($D_{hash}$) ++;
      else
         invertedIndex[word]. termFrequency ($D_{hash}$) ++;
      end if
   end if
end for
```

- Data-summary preprocessing. When the blockchain listens for a data-registration event, the DRC calls propressString() and stem() to traverse the Dsummary and complete the preprocessing. The preprocessing includes special character substitution, word separation and stemming, and the DRC builds the dictionary through these steps.
- Building the inverted index. The smart contract iterates through the dictionary and updates the inverted list *PostingList* with the inverted index table *invertedIndex* when the dictionary array *lexicon*[*i*] is not empty and not for retrieving stop words. The inverted list stores a list of data indexes as well as word frequencies. The inverted index table is in the form of key–value pairs, where the key is the word and the value is the inverted list corresponding to the word. Through the above steps, the smart contract establishes a keyword dictionary and an inverted index table corresponding to each of these words. In addition, due to the peculiarities of the permanent storage of blockchain data, the data-retrieval contract designs a list of stop words. When you want data related to a keyword to no longer be searchable, add the keyword to the table.

3.3.3. Collaborative Task Management and Collaboration-Incentive Allocation

To protect data privacy, the blockchain only stores a summary of the data, so the registration of the data by data nodes, i.e., Tp, is completely subjective, and it is difficult to assess the quality of their local data. In order to cope with this problem, we introduce the fuzzy comprehensive evaluation method into smart contracts to realise the quality evaluation of on-chain registration data. In order to realise the dynamic evaluation of the dataset quality, the evaluation factors are divided into initial static-evaluation factors and dynamic-behaviour-evaluation factors. For example, for supervised classification questions, set the evaluation factors as shown in Table 3.

Table 3. Factor sets of the fuzzy comprehensive evaluation.

Initial Static Factors	Dynamic Behaviour Factors
Data size: D_{size}	Number of tasks completed: T_{num}
Sample label completeness: D_{lable}	Completion time for this task: T_{time}
Node computing power: D_{node}	Node Credits: N_{credit}

The three initial evaluation factors and T_{num} all belong to a skewed large trapezoidal distribution, corresponding to the function shown in Equation (2), where x is the true value of the evaluation factor and $S(x)$ is the rating of the evaluation factor, and the constants a and b are determined by the properties of the task and the associated dataset, respectively:

$$S(x) = \begin{cases} 0, x < a \\ \frac{x-a}{x-b}, a \leq x \leq b \\ 1, b < x \end{cases} \tag{2}$$

T_{time} is affiliated with a small skewed trapezoidal distribution with the corresponding function shown in Equation (3), and t is the predefined model submission-time threshold for the collaborative task:

$$S(x) = \begin{cases} 1, x < t \\ \frac{2t-x}{t}, t \leq x \leq 2t \\ 0, 2t < x \end{cases} \tag{3}$$

N_{credit} is set as an adjustable constant with the update expression as in Equation (4). C is the current credit value, C_r is the task-completion credit reward value and C_p is the noncompletion credit penalty value. θ_i is the value of the pretrained model-verification metric for data node i, such as the model-classification accuracy, and θ is the model-verification metric threshold:

$$C = \begin{cases} C + Cr, \theta_i \geq \theta \\ C - Cp, \theta_i < \theta \end{cases} \tag{4}$$

The evaluation steps are shown in Algorithm 3, including initialising the data scores and updating the data scores. The smart contract calculates a composite score Dscore based on a predefined weight vector and evaluation factors, as shown in Equation (5). E_i is the evaluation factor, w_i is the corresponding weight and $D_{score} \in [0, 1]$, with higher scores representing the higher quality of the corresponding dataset:

$$D_{score} = \sum w_i \times E_i \tag{5}$$

In summary, the data-index construction and data-quality evaluation are performed through smart contracts, and the data registered by the data nodes to the blockchain are organised into the structure shown in Figure 4. Each piece of data in the trusted database is associated with a data owner and is highly structured in a way that creates an inverted index for easy data retrieval. A data-quality-evaluation mechanism encourages data nodes to upload high-value data, and records of dynamic updates to the data scores are permanently stored in the blockchain in the form of transactions.

Algorithm 3 Data evaluation (DEC)

```
1. Initialisation: when the blockchain has listened for data-registration event: $initScore$ = callInitialScore (get$D_{description}$ ($D_{hash}$));
2. updateDataScore(): require ($msg.sender_{role}$ == $Su$);
if the compute node completes the task as required then
    $ScoreSet\ [D_{hash}].nodeCredit{+}{+}$;
    $ScoreSet\ [D_{hash}].taskNumber{+}{+}$;
    $ScoreSet\ [D_{hash}].taskTime = T_{time}$;
else
    $ScoreSet\ [D_{hash}].nodeCredit$ -= 2;
end if
$D_{score} = initScore + \Sigma E_i \cdot \omega_i$;
```

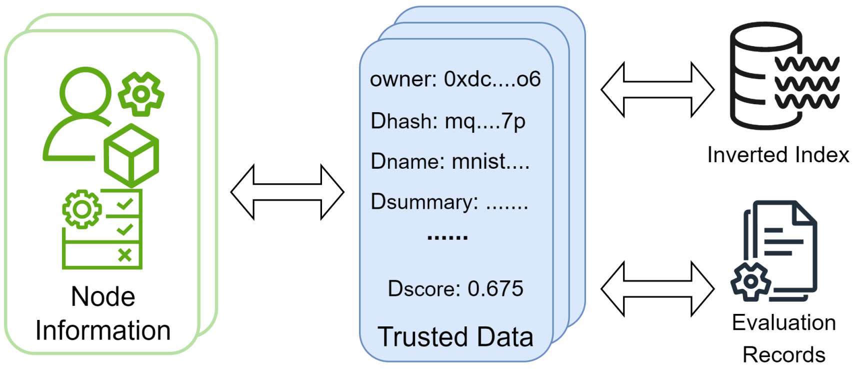


Figure 4. Structure of collaborative data on the blockchain.

3.3.4. Collaborative Task Management and Collaboration Incentive Allocation

Collaborative task management and incentive allocation are achieved by the TMC and TIC. The TMC records the collaborative tasks registered to the blockchain and controls the task process by changing the task status to ensure the trustworthiness and fairness of the tasks. The TIC incentivises Tp_s and Mv_s according to predefined rules and model-verification results. In the collaborative-learning mechanism, the task state is divided into: initialisation, formal publication, model-verification consensus and task end. Algorithm 4 shows the detailed steps of the task-management contract, Algorithm 5 introduces the incentive mechanism of the collaborative task and the details of the collaborative interactions at each stage of the collaborative-learning task are described below. In the collaborative task, there are m task participants and n model verifiers.

(a) Task initialisation. *To* calls taskRegister() to register the collaborative-learning task with the blockchain and pledge the task incentive, after which the smart contract initialises the task status and publishes the associated event. After task initialisation is completed, the TMC randomly assigns V_n model verifiers according to the task owner's requirements and determines whether sufficient task incentives have been pledged according to Equation (6). In Equation (6), T_{amount} is the incentive amount already pledged by *To* and k is the Mv reward factor, $k \in (0, 1]$.

$$T_{amount} \geq Tp_s.length \times T_{reward} \times V_n \times k \tag{6}$$

(b) Task formal publication. After the smart contract determines that *To* has pledged sufficient task incentives, the task is officially published and *To* provides a standard test set of collaborative tasks for all model verifiers. Tp_s develop pretrained models by using their local data and subsequently send the models to all designated model verifiers. The above interactions are transmitted encrypted via the transmit() function shown in Equation (7), which prevents the leakage of data and models related to the collaborative task and is stored in the blockchain in the form of transactions. In Equation (7), *from* is the address of the message publisher, *to* is the address of the message recipient, *key* is the public key of the message recipient and the data are stored by IPFS [30] to reduce the storage overhead of the blockchain:

$$transmit(from, to, key, IPFS(message)) \tag{7}$$

Algorithm 4 Collaborative task management (TMC)

```
1. taskRegister():
if $msg.sender$ has audited and this task is not registered then
    Update database: $TaskSet\ [T_{name}] = taskDesc\ (T_{name}, T_{summary}, T_{reward}, Tp_s, V_\theta, T_{secret})$;
    $TaskSet\ [T_{name}].status$ = initialisation;
else
    return false
end if
2. taskPublish():
if $T_{amount} \geq Tp_s.length * T_{reward} + Mv_s.length * T_{reward} * k$ then
    $TaskSet\ [T_{name}].status$ = formal publication;
    for each verifier $j$ in $Mv_s$ do
        Transmit ($To$, $j$, $key_j$, IPFS(validation set));
        Participants use private data to train their own local model;
    end for
    for each participant $i$ in $Tp_s$ and each verifier $j$ in $Mv_s$ do
        Transmit ($i$, $j$, $key_j$, IPFS($model_i$));
    end for
end if
3. verifyConsensus():
for each verifier $j$ in $Mv_s$ do
    $result_j$ = Hash ($acc_1, acc_2, \ldots, acc_m, secret_j$);
end for
The random verifier uploads the plaintext result – $accuracy$ []
$consensusFlag$ = true
for each verifier $j$ in $Mv_s$ do
    if Hash ($accuracy$ [], $secret_j$) $\neq result_j$ then
        $consensusFlag$ = false;
    end if
end for
if $consensusFlag$ = true then
    Go to Algorithm 5
    for each verifier $j$ in $Mv_s$ do
        Transmit ($j$, $To$, $key_{To}$, IPFS($verifiedmodels$));
    end for
else
    Go to (3);
end if
4. updateEvaluation(): require ($fundsToParticipants$ and $fundsToVerifiers$), go to Algorithm 1. (3) and Algorithm 2, $TaskSet\ [T_{name}].\ status$ = task end;
```

(c) Model-verification consensus and collaborative-task incentives. After receiving multiple pretrained models, model verifiers perform model verification by using the standard test set and upload the cryptographic verification results through the smart contract after completing all model verifications. The verification metric is the model-classification accuracy, and the cryptographic verification result of model verifier *j* is shown in Equation (8). Once all verifiers have completed uploading their encrypted verification results, the smart contract randomly assigns a verifier to upload the plaintext verification results and compares them to the encrypted verification results. If the comparison results are consistent, the smart contract declares a consensus on model verification, and *Mv* calls transmit() to send the validated pretrained model to *To* as an on-chain encrypted transmission. If the consensus is not reached, *Su* steps in and restarts a new round of model-validation consensus. Once the TMC determines that consensus has been reached, it will create the TIC for the task and transfer the *To* pledge incentive to that contract:

$$result_j = Hash(acc_1, acc_2, ..., acc_m, secret_j) \tag{8}$$

The incentive mechanism is shown in Equation (9), where the TIC incentivises task participant *i* by T_{reward} and the model accuracy weight, where θ_{count} is the number of models that satisfy the predefined verification threshold of the collaborative task, and the model verifier *j* incentive is determined by T_{reward} and incentive factor *k*. In addition, the TIC determines whether all collaborative nodes have been incentivised by setting incentive flag bits *fundsToParticipants* and *fundsToVerifiers*. The remaining funds are returned to *To* after the incentive is completed:

$$\begin{gathered} reward_{participants}[i] = \theta_{count} \times T_{reward} \times \frac{acc_i}{\Sigma accuracy} \\ reward_{participants}[j] = T_{reward} \times k \end{gathered} \tag{9}$$

(d) End of task. Consensus on model verification marks the completion of the main collaborative-learning step, and the subsequent allocation of incentives for collaboration is automatically performed by the smart contract according to predefined rules. At the same time, the DEC updates the data-quality evaluation, and the AMC updates the model-validator reputation value. Once the above steps are completed, the collaborative task ends and *To* completes global model development by fusing pretrained models. In summary, blockchain provides a trusted collaborative environment for collaborative-learning mechanisms, acting as a distributed ledger to record collaborative node interactions, and smart contracts to control the process of collaborative tasks through predefined rules, allowing for trusted and fair collaboration between collaborative nodes that do not fully trust each other.

Algorithm 5 Collaboration-incentive allocation (TIC)

```
1. getTaskInfo(): $I_{info}$ = getTaskInfo ($T_{name}$), fundsToParticipants = false, fundsToVerifiers = false;
2. nodeIncent():
for each participant $i$ in $Tp_s$ do
    if $accuracy[i] \geq \theta$ and fundsToParticipants = false then
        $Tp_i$.transfer($\theta_{count}$ * $T_{reward}$ * $\frac{acc_i}{\Sigma acc}$);
    end if
end for
fundsToParticipants = true;
for each verifier $j$ in $Mv_s$ do
    $Mv_j$.transfer($T_{reward}$ * $k$);
end for
fundsToVerifiers = true;
3. refund(): require (fundsToParticipants and fundsToVerifiers), To. transfer(address (this. balance));
```

3.4. Model-Fusion Method Based on Feature Fusion

Based on the above blockchain smart contract collaboration architecture and transfer learning [31], we propose a model-fusion method based on feature fusion to enable collaborative model training to aggregate dispersed data resources. The method consists of the following two steps:

1. Submodel pretraining: data node *i* ($i \in [1, n]$) develops a pretrained model *i* by using its local data.

2. Model fusion and fine tuning: The task-publishing node uses its local data to perform parallel-feature extraction from n pretrained submodels and then uses an attention mechanism to fuse the extracted features from the submodels to generate global features, which are finally used as the input to the classifier to obtain the global model.

Figure 5 depicts the global-model-development process for the collaborative-learning mechanism proposed in this paper, with steps 5 and 8 corresponding to the submodel pretraining, model fusion and fine-tuning steps, respectively. Figure 6 illustrates the model-fusion architecture. As shown in this architecture, in order to develop a global model, features from all pretrained submodels need to be integrated to generate global features. Define the global model input as x, the ith submodel feature extraction layer as a function f_i, E_i as the features extracted by submodel i, F as the feature-fusion operator and V as the global features. Then, the attention-based feature-fusion process can be formulated by Equations (10) and (11), where $E_i \in \mathbb{R}^{C\times H\times W}$, $E \in \mathbb{R}^{n\times C\times H\times W}$ and $V \in \mathbb{R}^{C\times H\times W}$:

$$\begin{cases} E_i = f_i(x) \\ E = F_{concat}(E_1, E_2, \ldots, E_n) \end{cases} \tag{10}$$

$$V = F_{attention}\left(E, f_1(x), f_2(x), \ldots, f_n(x)\right) \tag{11}$$

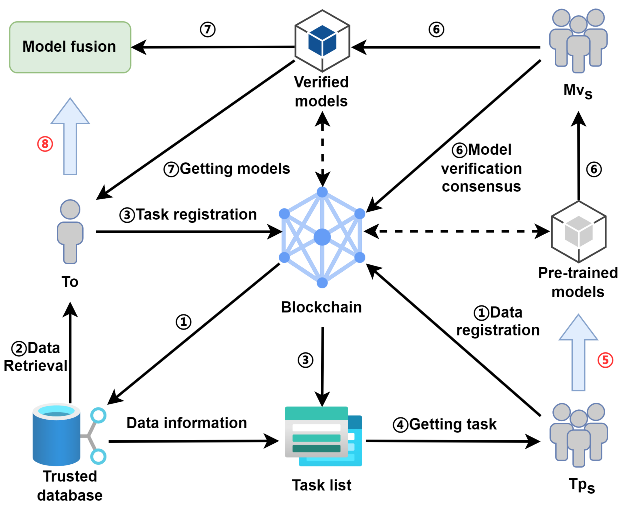


Figure 5. The process of collaborative training of the model based on blockchain.

To elaborate on $F_{attention}$, n is taken to be 3, i.e., 3 pretrained submodels are fused, as shown in the corresponding framework for the Attention Layer in Figure 6. Naturally, the present architecture can also be very easily extended to scenarios where n is greater than 3. The merged feature map E contains feature information extracted from all submodels, and the global feature information is embedded in the vector T ($T \in \mathbb{R}^{C\times 1\times 1}$) by using global average pooling, as shown in Equation (12). After the above steps are completed, the vector T is fed to the fully connected layer fc, which adaptively fuses the features further. In addition, to reduce the model computation, the number of fc neurons is set to d ($d \leq C$):

$$T = AvgPool(E) = \frac{\sum_{i=1}^{H}\sum_{j=1}^{W} E(i,j)}{H \times W} \tag{12}$$

$$D = fc(T) \tag{13}$$

The global feature information is further extracted by vector D. Subsequently, a soft attention mechanism is introduced in order to calculate the fusion weights of the features extracted by the different submodels. Corresponding to the three pretrained submodels, the fully connected layer fc connects each of the three fully connected layers, i.e., the three branches of the model: fc_1, fc_2, fc_3. Finally, the output vectors of the different fully connected layers are normalised by the softmax function to generate the feature attention vectors $w1$, $w2$ and $w3$. The number of neurons in *fc_1*, *fc_2* and *fc_3* are all set to C to recover the vector dimension, $w1, w2$ and $w3 \in \mathbb{R}^{C\times1\times1}$:

$$(w1, w2, w3) = softmax(fc\_1(D), fc\_2(D), fc\_3(D)) \tag{14}$$

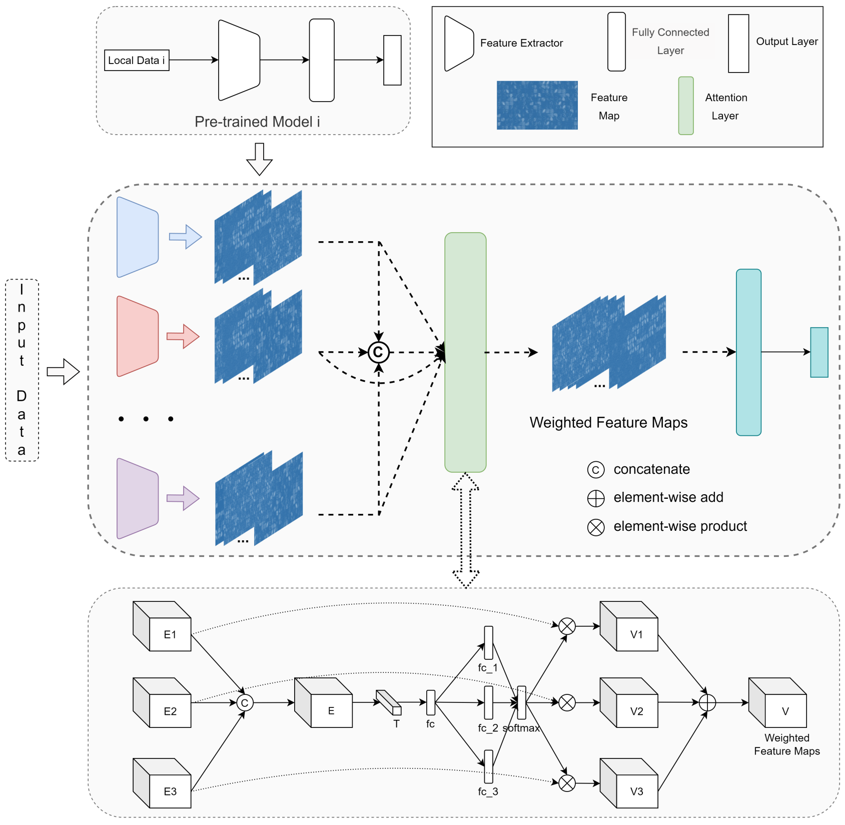


Figure 6. Model fusion and fine tuning: feature-fusion method based on the attention mechanism.

In the above calculation step, the weight vectors of the features extracted by the different submodels are obtained through the application of feature adaptive fusion and soft attention, and then the global features V can be calculated from Equations (15) and (16), $V_i, V \in \mathbb{R}^{C\times H\times W}$.:

$$V_i = w_i \times E_i(\sum_{i=1}^{n} w_i = 1) \tag{15}$$

$$V = V_1 + V_2 + V_3 \tag{16}$$

In summary, the proposed method models the association between features extracted from different submodels explicitly based on an attention mechanism and provides accurate guidance for model fusion through the adaptive calculation of feature weights.

4. Formal Security Analysis of Smart Contracts

Smart contracts act as automated protocols with inherent execution conditions and execution logic that define and guide distributed collaboration in a trustworthy and fair manner. The security of smart contracts is crucial for the application of the method proposed in this paper in collaborative learning. We use Coloured Petri Nets (CPNs) [32] to implement the formal modelling and verification of smart contracts to prove that they satisfy the expected trustworthiness attributes. In this section, attributes are modelled for smart contracts related to collaborative-learning mechanisms, and the hierarchical CPN model is constructed. We then describe the design of the formal model and verify the degree to which its attributes are satisfied by using a simulation, state space and Model Checking.

4.1. Preliminary

4.1.1. Formal Concepts for CPN

The CPN can be described and defined by a nine-tuple, i.e., $CPN = (P, T, A, \sum, V, C, G, E, I)$. P is the Place Set, T is the Transition Set and A is the Arc Set. The set of the three forms the CPN model infrastructure and satisfies the definition $A \subseteq P \times T \cup T \times P$. $\sum$ is the colour set of the CPN model, defining the data structure and type of the token in the model. V is the Variable Set, any set of variables $v \in \mathrm{V}$ in the model satisfies the definition Type[v] $\in \sum$. C is a colour set definition function that specifies the colour set for the token contained in each place of the model. G is the guard function of the transition, defining the defensive expression of the transition, whose return value is of Boolean type and is used to determine whether the transition can be fired or not. E is an arc expression function that specifies an expression for each directed arc in the model. I is an initialisation function that specifies the initial state of the model.

4.1.2. CPN-Based Formal Modelling and Verification

Based on the theory of Coloured Petri Nets and the CPN Tools [33], the simulation, state space analysis and Model Checking of CPN models can be achieved, and the formal verification of CPN models can be completed by the above methods. The simulation execution implements interactions with the model and automatic model-simulation execution. The state space tool automatically generates a reachable graph of all states of the model, based on which the ASK-CTL and CPN ML [34] languages are used to implement Model Checking and determine how well the model satisfies the desired attributes.

4.2. Attribute Modelling and the Top-Layer Model Design

4.2.1. Modelling Smart Contract Attributes

Smart contracts related to collaborative-learning mechanisms are defined as three types of functional modules that implement role-authority management, trusted database construction and collaborative task management and incentive allocation. Based on the above-mentioned functionalities and security guarantees, the smart contract model should satisfy the following attributes in order to ensure its trustworthiness:

Attribute 1: Collaborative nodes must be authorised by the smart contract to participate in collaborative learning. The smart contract assigns different authorities to collaborative nodes according to their roles and can immediately restrict their collaboration authority when they behave maliciously.

Attribute 2: collaborative nodes must meet the conditions specified by the smart contract restriction function to perform on-chain operations.

Attribute 3: consensus on model verification requires all model verifiers to participate, and the task-management contract creates incentive contracts for collaborative tasks only when consensus is reached.

Attribute 4: The smart contract can initiate trust arbitration for model verifiers. If more than half of the task participants believe that they are suspected of malicious behaviour, the smart contract shall remove their model-verifier role authority.

4.2.2. Top-Layer Model Design

We use a hierarchical modelling approach to build a two-layer CPN model for smart contracts, with the top-layer model shown in Figure 7. The top-layer model models the behaviour of each of the four aspects of smart contracts, namely role-authority management, role-authority validation, model-validation consensus and model-validator trust arbitration, represented by four submodules: the Authority layer, Validation layer, Consensus layer and Arbitration layer. The top-layer model consists of nine places, with *admin* denoting *Su*, which is responsible for authorising collaborator nodes, creating model-verifier trust arbitration and overseeing the model-verification consensus. *Co_Info* and *Verifiers_Info* denote the collaborators (*To* and Tp_s) to be authorised and the model verifiers to be authorised, respectively. *Co_nodes* and *V_nodes* correspond to the list of authorised collaborators and authorised model verifiers, respectively. The Validation layer models the smart contract and requires the restriction function, so *Collaborators* and *Verifiers* denote the collaborators and model verifiers operating in the request chain, respectively, and *Collaborators_v* and *Verifiers_v* denote the collaborators and model verifiers of operations in the request chain, respectively.

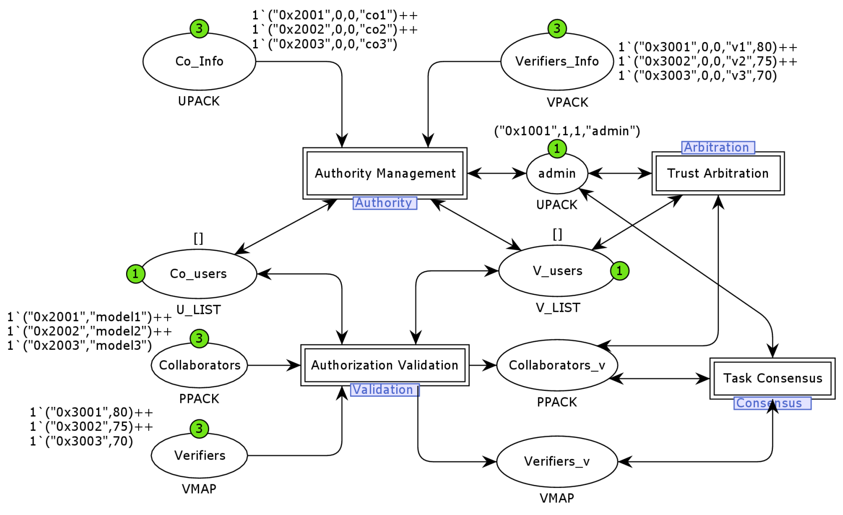


Figure 7. The top layer of CPN model.

4.3. Design and Analysis of Authority Layer and Validation Layer

4.3.1. Formal Modelling

The Authority layer model is the bottom implementation of the authority-management substitution transition, which models the role-authority-management function. As shown in Figure 8, the model contains a total of 17 places and 10 transitions, with five of the port places connected to the top-layer CPN model. The places *UInfo* and *VInfo* denote the collaborators and model verifiers to be authorised, respectively, and *C_list* and *V_list* are the lists of collaborators and model verifiers that have been authorised, respectively. Collaborator permissions are modelled as a quadruplet (UAdd, URole, UStatus and UInfo) with the collaborating node's Ethernet address, node role, node-identity status and node information overview, respectively. The model verifier authority is modelled as a five-tuple (UAdd, URole, UStatus, UInfo and V_Credit), with the first four items of authority information being the same as the collaborator and the fifth item being the model-verifier credibility value. Therefore, we will introduce the places and transitions of the Authority layer in terms of the model-verifier authority-management process, without going into the rest of the content. The change *Init_V* models the collaborative node information-initialisation behaviour, modifying the node-identity status from pending review to legal, and *V_Auth* models the smart contract assignment of node role-authority behaviour, modifying the node role from pending assignment to model verifier. The places *V_Nodes* and *Co_Verifier* denote model verifiers before and after the role-authority assignment, respectively. The place *v_times* denotes the number of times the smart contract presets role-authority assignment to limit the supervisor's authority assignment behaviour. The transition *V_As* adds model verifiers that have been assigned collaborative node role authority to the list *V_list* and also records the corresponding authority information to the place *v_node*. When a model verifier behaves suspiciously, the *Su* can restrict its role authority through the AMC. The transitions *Cancel_V* and *Deau_V* model the behaviour of the AMC restricting the authority of a model verifier, and the places *cancel_v* and *V_Malicious* denote a malicious model verifier with its corresponding suspicious behaviour, respectively.

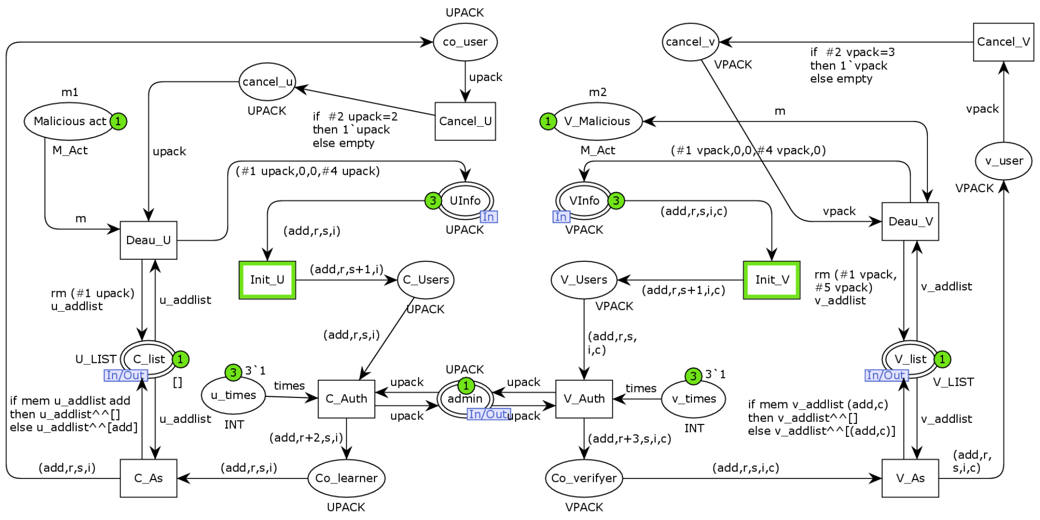


Figure 8. The Authority layer of CPN model.

The Validation layer is the bottom implementation of the Authorisation Validation substitution transition, which models the role-authority-validation function in smart contracts. As shown in Figure 9, the transition *Validation_C* models the collaborator-authority-validation behaviour, which is achieved by determining whether a collaborator belongs to the list of legitimate collaborators. The place of *require_1* indicates the collaborator request, and the place of *right_1* indicates the result of the smart contract authority-restriction function, which returns "access" if it passes the authority verification; otherwise, it returns "no acces". The transitions *Validation_C* and *Validation_U* together represent the authority-validation behaviour, where *Validation_C* determines whether the model verifier requesting an on-chain operation belongs to the list of legitimate verifiers, *Validation_U* determines whether the corresponding model verifier meets the credibility threshold requirements and the places *require_2* and *right_2* represent the results returned by the model-verifier on-chain operation request and the smart contract authority-restriction function, respectively.

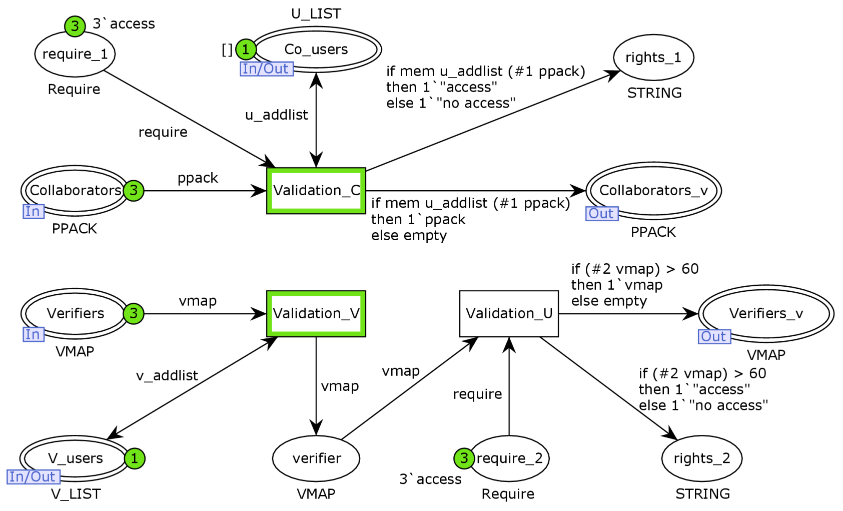


Figure 9. The Validation layer of CPN model.

4.3.2. Formal Verification

To verify the degree of model-attribute satisfaction, based on the Authority and Validation layer models, we set the initial marking of the top-layer model as M_0 and use the model verifier v1 as an example to execute the authority-management and verification process by using the simulation tool as follows:

Step 1 M_0[*Init_V*>M_1, *Init_V* transition fired and contract status is converted from M_0 to M_1. At this point, E(*VInfo*, *Init_2*) <add, r, s, i, c>= 1′(“0x3001”, 0, 0, “v1”, 80). That is, the arc pointing to the *Init_V* transition binds the authority information of the v1 to be authorised. After the *Init_V* transition is fired, the v1 status is updated to the legal status.

Step 2 M_1[*V_Auth*>M_2, *V_Auth* transition fired and contract status is converted from M_1 to M_2. At this point, E(*V_Nodes*, *V_Auth*) <add, r, s, i, c>= 1′(“0x3001”, 0, 1, “v1”, 80), E(*v_times*, *V_Auth*) <times>= 1, E(*admin*, *V_Auth*) <upack>= 1′(“0x1001”, 1, 1, “admin”). After the *V_Auth* transition is fired, the contract assigns the model-verifier role authority to v1.

Step 3 M_2[*V_As*>M_3, *V_As* transition fired and contract status is converted from M_2 to M_3. At this point, E(*co_verifiers*, *V_As*) <add, r, s, i, c>= 1′(“0x3001”, 3, 1, “v1”, 80). After the *V_As* transition is fired, the contract adds v1 to the list of legitimate model verifiers *V_List*.

Step 4 M_3[*Validation_V*>M_4, *Validation_V* transition fired and contract status is converted from M_3 to M_4. At this point, E(*Verifiers*, *Validation_V*) >vmap>= 1′(“0x3001”, 80), E(*V_nodes*, *Validation_V*) <v_addlist>= 1′[(“0x3001”, 3, 1, “v1”, 80)]. It is used to determine whether or not v1 has legal-role authority.

Step 5 M_4[*Validation_U*>M_5, *Validation_U* transition fired and contract status is converted from M_4 to M_5. At this point, E(*verifier*, *Validation_U)* <vmap>= 1′(“0x3001”, 80), E(*V_nodes*, *Validation_U)* <require_2>= “access”. That is, the two arcs pointing to the *Validation_U* transition bind the v1 operating on the application chain to the application request, respectively. After the *Validation_U* transition is fired, the contract determines whether the corresponding credibility value of v1 is greater than the required threshold. Here, the threshold is met, and then v1 can perform the on-chain operation it requested.

Step 6 M_5[*Cancel_V*>M_6, *Cancel_U* transition fired and contract status is converted from M_5 to M_6. At this point, E(*v_node*, *Cancel_V*) <vpack> = 1′(“0x3001”, 3, 1, “v1”, 80). That is, the arc pointing to the transition *Cancel_V* binds information about the model-verifier authority for possible malicious behaviour.

Step 7 M_6[*Deau_V*>M_6, *Deau_U* transition fired and contract status is converted from M_5 to M_6. At this point, E(*V_Malicious*, *Deau_V*) <m>= m2, E(*cancel_v*, *Deau_V*) <vpack>= 1′(“0x3001”, 3, 1, “v1”, 80). That is, the contract confirms the presence of malicious behaviour by verifier v1. After the *Deau_V* transition is fired, the contract removes v1 from the list of legitimate verifiers and initialises its authority information to (“0x3001”, 0, 0, “v1”, 0).

In the above steps, steps 1 to 3 simulate the process of the contract assigning the verifier authority to model verifier v1, and steps 6 and 7 simulate the process of the contract restricting the authority of v1 in the event of suspected malicious behaviour. The process of contract authorisation and the restriction of permissions is atomic and requires supervisor permissions when role permissions are assigned. The first attribute is therefore safeguarded. Steps 4 and 5 simulate the contract authority-verification function. From the top-layer model design and the above steps, it is clear that authority verification is a precondition for nodes to apply for on-chain operations and that this verification process is not influenced by external factors. Therefore, the second attribute is guaranteed.

4.4. Design and Analysis of Consensus Layer

4.4.1. Formal Modelling

The Consensus layer is the bottom implementation of the Task-Consensus substitution transition, which models the consensus mechanism for model verification in collaborative-learning mechanisms. As shown in Figure 10, the place *Collaborators_v* indicates the task participants who have passed the authority verification, and the place *Verifiers_v* indicates the model verifiers who have passed the authority verification. The transition *Start Verification* indicates that the task participant has completed local pretraining model development and is ready to send the model to the model verifier to start the model-verification consensus, and the places *Init Flag* and *start consensus* indicate that the smart contract initialises the model-verification consensus flag bit. The transition *Send Model* indicates model transmission, the places *participant* and *gas* model the chain-transmission process and the place *model* indicates that all model verifiers have received the model. The transition *Distribute Key* models the encryption upload process, the place *secret* represents the encrypted random numbers assigned to the different model verifiers and place *verifier_s* stores the model verifiers and their corresponding random numbers. The transition *verify* models the smart contract model-verification-consensus process, and the place *accuracy* represents the plaintext verification results uploaded by a randomly assigned model verifier. The place *match* records the number of results that have been matched, and the place *mismatch* records the addresses of model verifiers that do not agree with the plaintext results. The transitions *consensus* and *No consensus* indicate consensus reached and consensus failed, respectively.

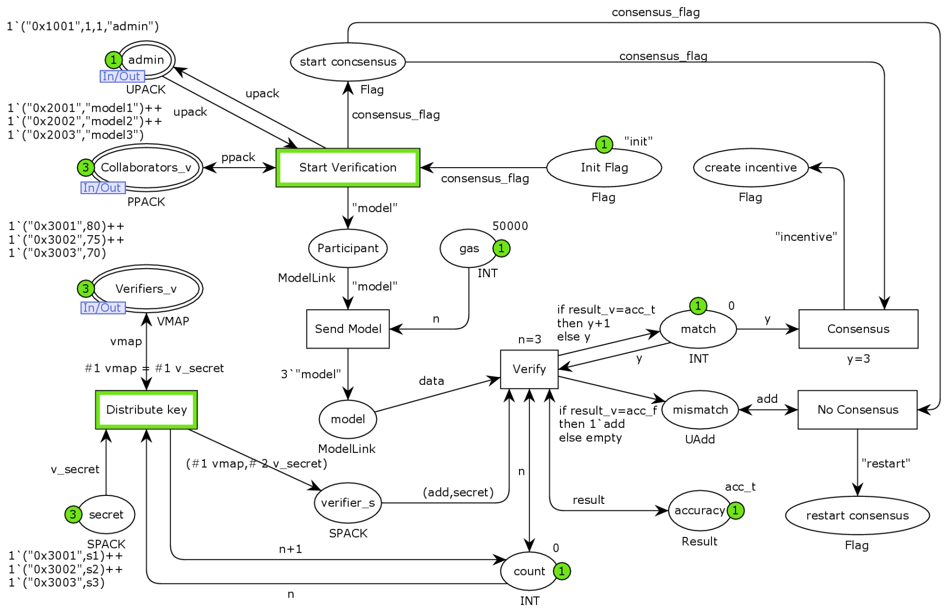


Figure 10. The Consensus layer of CPN model.

4.4.2. Formal Verification

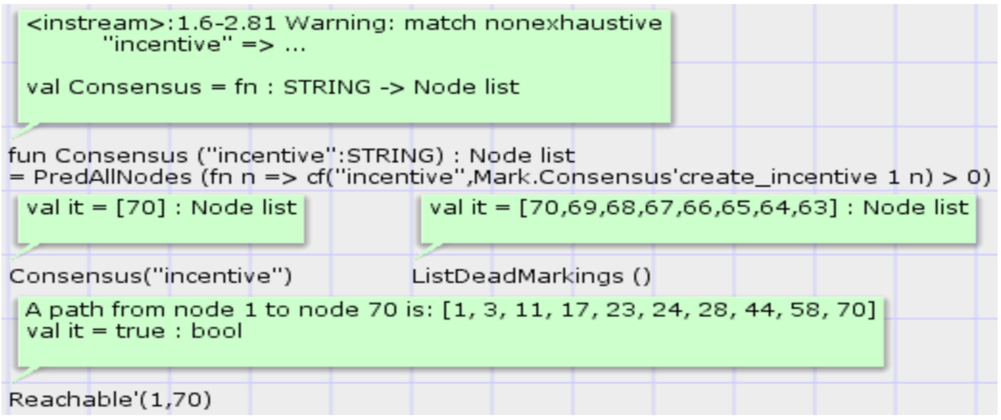
Let the model status shown in Figure 10 be M_i, and the steps to perform a model-validation consensus by using the simulation tool are shown in Table 4. Steps 6 to 9 indicate that the smart contract will only judge consensus reached if all verifiers participate in the model-verification consensus and the cryptographic verification results are consistent. Thus, the third property is guaranteed. To further illustrate the correctness of the model and simulation results, the Consensus layer is analysed next and verified by using the state space tool and Model Checking. Execute the following status formula:

```
fun Consensus ("incentive": STRING) : Node list =
PredAllNodes (fn n =>cf(''incentive'', Mark.Consensus'create_incentive 1 n) >0);
    Consensus("incentive");
```

The above state formula is executed to find the state space node corresponding to the model status after the consensus transition is fired, and the function is executed to return node 70. The path query "Reachable' (1, 70)" finds the state space node path from the initial marking to the model-verification consensus, and the result shows (1, 3, 11, 18, 23, 24, 28, 44, 58, 70). Therefore, after the transition consensus is fired, the model will only produce one marking, consistent with the simulation execution results. Figure 11 shows the results of the execution of the above state formula and query statement.

Table 4. The transition enables sequence of model-verification consensus.

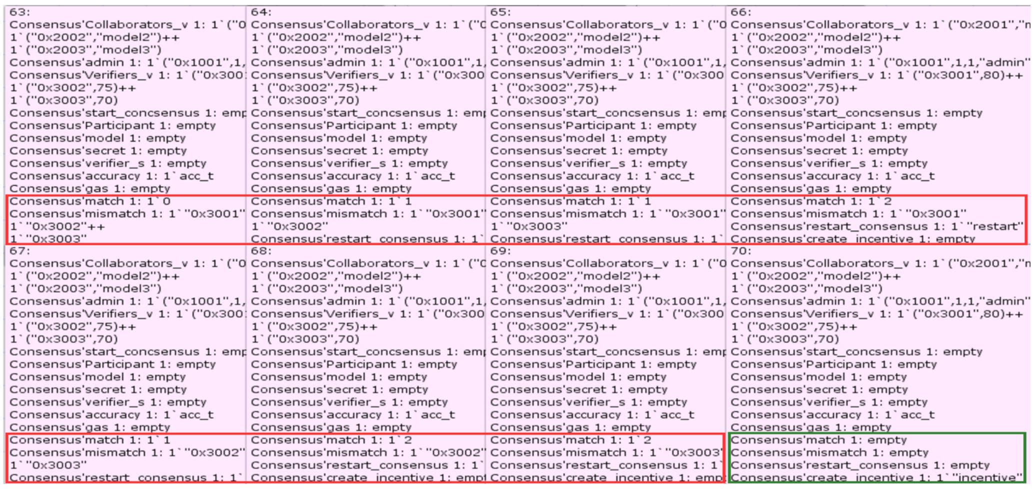
Step	Simulations
1	M_i[*Start Verification*>M_{i+1}, smart contract initialises consensus flag.
2	M_{i+1}[*Send Model*>M_{i+2}, *Tp* sends pretrained models to all model verifiers.
3	M_{i+2}[*Send Model*>M_{i+3}, smart contracts specify the cryptographic number for model verifier v1.
4	M_{i+3}[*Send Model*>M_{i+4}, smart contracts specify the cryptographic number for model verifier v2.
5	M_{i+4}[*Send Model*>M_{i+5}, smart contracts specify the cryptographic number for model verifier v3.
6	M_{i+5}[*Verify*>M_{i+6}, smart contract verifies the encryption verification result of v1.
7	M_{i+6}[*Verify*>M_{i+7}, smart contract verifies the encryption verification result of v2.
8	M_{i+7}[*Verify*>M_{i+8}, smart contract verifies the encryption verification result of v3.
9	M_{i+8}[*Consensus*>M_{i+9}, smart contract determines that a model-verification consensus has been reached, modifies the consensus flag bit and creates TIC.



```
<instream>:1.6-2.81 Warning: match nonexhaustive
          "incentive" => ...

val Consensus = fn : STRING -> Node list

fun Consensus ("incentive":STRING) : Node list
= PredAllNodes (fn n => cf("incentive",Mark.Consensus'create_incentive 1 n) > 0)

val it = [70] : Node list
Consensus("incentive")

val it = [70,69,68,67,66,65,64,63] : Node list
ListDeadMarkings ()

A path from node 1 to node 70 is: [1, 3, 11, 17, 23, 24, 28, 44, 58, 70]
val it = true : bool
Reachable'(1,70)
```

Figure 11. The execution results state formula and query statements of Consensus layer.

To further illustrate the status of the model when a consensus on model verification is not reached, a status query statement "ListDeadMarkings()" is used to list all the dead marks of the model, as shown in Figure 11. The results show that in addition to state node 70, state nodes 63 to 69 indicate the model status when a model-verification consensus has not been reached. As shown in Figure 12, we listed the eight dead markings of the model through the state space tool, and the results show that state nodes 63 to 69 are all in the status of not reaching consensus on model verification, at which point the smart contract records the addresses of model verifiers with inconsistent verification results (the token values in the place *mismatch*).



```
63:
Consensus'Collaborators_v 1: 1`("0
1`("0x2002","model2")++
1`("0x2003","model3")
Consensus'admin 1: 1`("0x1001",1,
Consensus'Verifiers_v 1: 1`("0x300
1`("0x3002",75)++
1`("0x3003",70)
Consensus'start_concsensus 1: emp
Consensus'Participant 1: empty
Consensus'model 1: empty
Consensus'secret 1: empty
Consensus'verifier_s 1: empty
Consensus'accuracy 1: 1`acc_t
Consensus'gas 1: empty
Consensus'match 1: 1`0
Consensus'mismatch 1: 1`"0x3001"
1`"0x3002"++
1`"0x3003"
```

```
64:
Consensus'Collaborators_v 1: 1`("0
1`("0x2002","model2")++
1`("0x2003","model3")
Consensus'admin 1: 1`("0x1001",1,
Consensus'Verifiers_v 1: 1`("0x300
1`("0x3002",75)++
1`("0x3003",70)
Consensus'start_concsensus 1: emp
Consensus'Participant 1: empty
Consensus'model 1: empty
Consensus'secret 1: empty
Consensus'verifier_s 1: empty
Consensus'accuracy 1: 1`acc_t
Consensus'gas 1: empty
Consensus'match 1: 1`1
Consensus'mismatch 1: 1`"0x3001"
1`"0x3002"
Consensus'restart_consensus 1: 1`
```

```
65:
Consensus'Collaborators_v 1: 1`("0
1`("0x2002","model2")++
1`("0x2003","model3")
Consensus'admin 1: 1`("0x1001",1,
Consensus'Verifiers_v 1: 1`("0x300
1`("0x3002",75)++
1`("0x3003",70)
Consensus'start_concsensus 1: emp
Consensus'Participant 1: empty
Consensus'model 1: empty
Consensus'secret 1: empty
Consensus'verifier_s 1: empty
Consensus'accuracy 1: 1`acc_t
Consensus'gas 1: empty
Consensus'match 1: 1`1
Consensus'mismatch 1: 1`"0x3001"
1`"0x3003"
Consensus'restart_consensus 1: 1`
```

```
66:
Consensus'Collaborators_v 1: 1`("0x2001","n
1`("0x2002","model2")++
1`("0x2003","model3")
Consensus'admin 1: 1`("0x1001",1,1,"admin"
Consensus'Verifiers_v 1: 1`("0x3001",80)++
1`("0x3002",75)++
1`("0x3003",70)
Consensus'start_concsensus 1: empty
Consensus'Participant 1: empty
Consensus'model 1: empty
Consensus'secret 1: empty
Consensus'verifier_s 1: empty
Consensus'accuracy 1: 1`acc_t
Consensus'gas 1: empty
Consensus'match 1: 1`2
Consensus'mismatch 1: 1`"0x3001"
Consensus'restart_consensus 1: 1`"restart"
Consensus'create_incentive 1: empty
```

```
67:
Consensus'Collaborators_v 1: 1`("0
1`("0x2002","model2")++
1`("0x2003","model3")
Consensus'admin 1: 1`("0x1001",1,
Consensus'Verifiers_v 1: 1`("0x300
1`("0x3002",75)++
1`("0x3003",70)
Consensus'start_concsensus 1: emp
Consensus'Participant 1: empty
Consensus'model 1: empty
Consensus'secret 1: empty
Consensus'verifier_s 1: empty
Consensus'accuracy 1: 1`acc_t
Consensus'gas 1: empty
Consensus'match 1: 1`1
Consensus'mismatch 1: 1`"0x3002"
1`"0x3003"
Consensus'restart_consensus 1: 1`
```

```
68:
Consensus'Collaborators_v 1: 1`("0
1`("0x2002","model2")++
1`("0x2003","model3")
Consensus'admin 1: 1`("0x1001",1,
Consensus'Verifiers_v 1: 1`("0x300
1`("0x3002",75)++
1`("0x3003",70)
Consensus'start_concsensus 1: emp
Consensus'Participant 1: empty
Consensus'model 1: empty
Consensus'secret 1: empty
Consensus'verifier_s 1: empty
Consensus'accuracy 1: 1`acc_t
Consensus'gas 1: empty
Consensus'match 1: 1`2
Consensus'mismatch 1: 1`"0x3002"
Consensus'restart_consensus 1: 1`
Consensus'create_incentive 1: empt
```

```
69:
Consensus'Collaborators_v 1: 1`("0
1`("0x2002","model2")++
1`("0x2003","model3")
Consensus'admin 1: 1`("0x1001",1,
Consensus'Verifiers_v 1: 1`("0x300
1`("0x3002",75)++
1`("0x3003",70)
Consensus'start_concsensus 1: emp
Consensus'Participant 1: empty
Consensus'model 1: empty
Consensus'secret 1: empty
Consensus'verifier_s 1: empty
Consensus'accuracy 1: 1`acc_t
Consensus'gas 1: empty
Consensus'match 1: 1`2
Consensus'mismatch 1: 1`"0x3003"
Consensus'restart_consensus 1: 1`
Consensus'create_incentive 1: emp
```

```
70:
Consensus'Collaborators_v 1: 1`("0x2001","n
1`("0x2002","model2")++
1`("0x2003","model3")
Consensus'admin 1: 1`("0x1001",1,1,"admin"
Consensus'Verifiers_v 1: 1`("0x3001",80)++
1`("0x3002",75)++
1`("0x3003",70)
Consensus'start_concsensus 1: empty
Consensus'Participant 1: empty
Consensus'model 1: empty
Consensus'secret 1: empty
Consensus'verifier_s 1: empty
Consensus'accuracy 1: 1`acc_t
Consensus'gas 1: empty
Consensus'match 1: empty
Consensus'mismatch 1: empty
Consensus'restart_consensus 1: empty
Consensus'create_incentive 1: 1`"incentive"
```

Figure 12. The results of smart contract model verification.

4.5. Design and Analysis of Arbitration Layer

4.5.1. Formal Modelling

The Arbitration layer model is the bottom implementation of the trust arbitration substitution transition, which models the act of a smart contract initiating trust arbitration on a model verifier. As shown in Figure 13, the model has three port places connected to the top-layer model, where the place *Collaborator_v* denotes the task participants involved in trust arbitration, the place *admin* denotes the supervisor who created the arbitration and the place *V_Nodes* denotes the list of legitimate model verifiers. The transition *Create Arbitration* indicates that the supervisor creates trust arbitration for the model verifier with suspicious behaviour via the smart contract, and the place *s_verifiers* indicates that verifier. The place *initiate* indicates the act of a task participant proposing trust arbitration, the task participants join trust arbitration through the transition *Join Arbitration* and the transition *verification* indicates the verification and confirmation of the suspicious behaviour. The places *confidence vote* and *suspicious behaviour* indicate that all task participants have completed the suspicious-behaviour confirmation and subsequently start the trust arbitration vote through variation voting. Furthermore, we model a counter via place *limit* to ensure that each participant has only one chance to vote. The places *distrust* and *trust* indicate that the participants do not trust or trust the model verifier, and the transition *Cancel_Au* models the act of revoking authority. If only half or less of the participants believe that the model verifier is behaving suspiciously, the smart contract reduces the model verifier credibility value based on the number of untrusted votes, and the transitions *Reduce_C* and *Update* model the behaviour, with the place *temp* indicating that the model verifier information is temporarily stored.

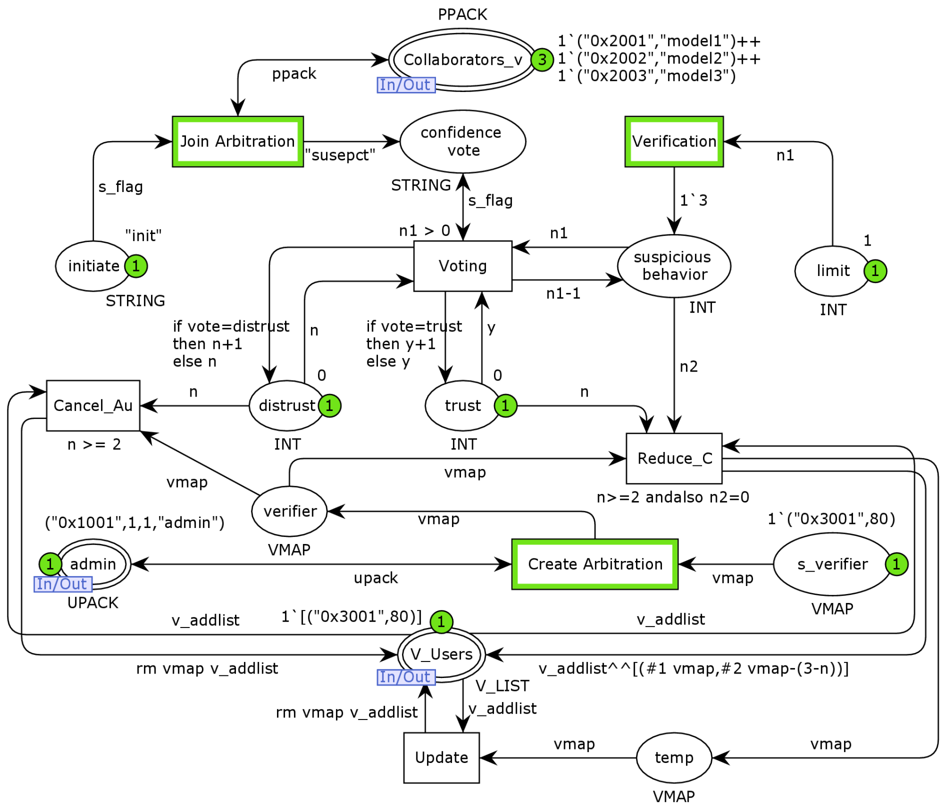


Figure 13. The Arbitration layer of CPN model.

4.5.2. Formal Verification

Let the model status shown in Figure 13 be M_i, and the steps to perform the model-verifier trust arbitration by using the simulation tool are shown in Table 5. According to the corresponding design of the Arbitration layer, in addition to the above simulation results, if there are task participants who choose not to trust the model verifier, the smart contract will reduce the reputation value of the model verifier or remove its verifier role authority based on the trust arbitration results. The state space tool gives the simulation results shown in Table 5 corresponding to the model state nodes as 33. We use the query statement "Reachable' (1, 33)" to find the state space node path from the initial marking to the completion of trust arbitration, and the result shows (1, 3, 7, 10, 15, 18, 27, 31, 33). We then used the query statement "ListDeadMarkings()" to list all the dead markings of the model and the results showed (25, 28, 29, 32, 33). From the previous analysis, it can be seen that the four dead markings except state node 33 should be the case where there are task participants who do not trust the model verifier, and to verify the correctness of the model in this case, the following CPN ML status formula is executed:

fun Cancel (v_addlist: V_LIST) : Node list = PredAllNodes (fn n =>cf([], Mark. Arbitration'V_Nodes 1 n) >0);
Cancel([]);
fun Reduce(v_addlist:V_LIST) : Node list = PredAllNodes (fn n =>cf([("0x3001", 79)], Mark. Arbitration'V_Nodes 1 n) >0);
Reduce([("0x3001", 79)]);

The Cancel function indicates that more than half of the task participants believe that the model verifier is behaving suspiciously, and the Reduce function indicates that only half or less of the task participants choose not to trust the model verifier. Figure 14 shows the results of the execution of the above query statement and status formula, with the Cancel function returning a list of nodes (25, 28, 29) and the Reduce function returning a list of nodes (32), showing that there were no unexpected error dead markings.

Table 5. The transition-enabled sequence of model-verifier trust arbitration.

Step	Simulations
1	M_i[*Create Arbitration* $\rightarrow M_{i+1}$], smart contract creates trust arbitration.
2	M_{i+1}[*Join Arbitration* $\rightarrow M_{i+2}$], task participants participate in the vote of confidence in arbitration.
3	M_{i+2}[*Verification* $\rightarrow M_{i+3}$], task participants confirm suspicious behaviour of the model verifier.
4	M_{i+3}[*Voting* $\rightarrow M_{i+4}$], task participant 1 selects trust model verifier.
5	M_{i+4}[*Voting* $\rightarrow M_{i+5}$], task participant 2 selects trust model verifier.
6	M_{i+5}[*Voting* $\rightarrow M_{i+6}$], task participant 3 selects trust model verifier.
7	M_{i+6}[*Reduce_C* $\rightarrow M_{i+7}$], smart contract adds model verifier with updated credibility value to list.
8	M_{i+7}[*Update* $\rightarrow M_{i+8}$], smart contracts remove old model verifier information from the list.

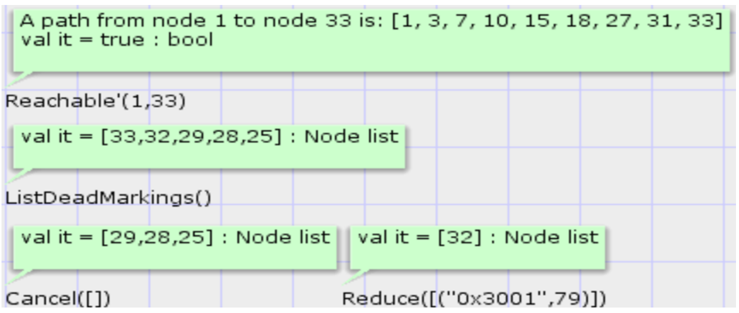
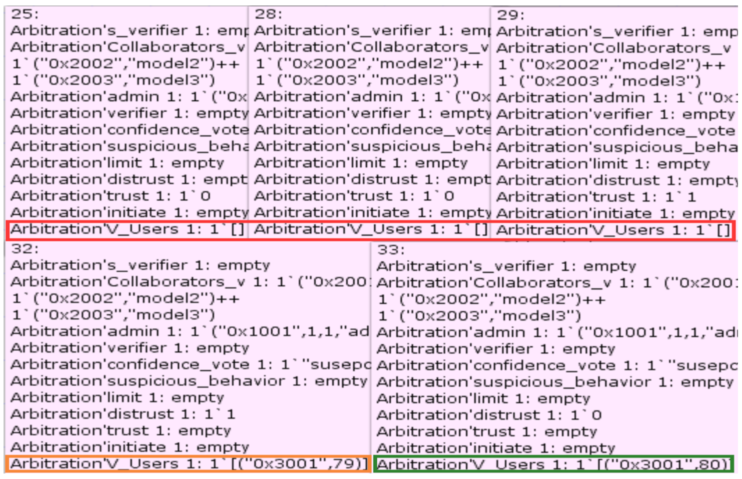

```
A path from node 1 to node 33 is: [1, 3, 7, 10, 15, 18, 27, 31, 33]
val it = true : bool
Reachable'(1,33)
val it = [33,32,29,28,25] : Node list
ListDeadMarkings()
val it = [29,28,25] : Node list
Cancel([])
val it = [32] : Node list
Reduce([("0x3001",79)])
```

Figure 14. The execution results of status formula and query statement of Arbitration layer.

As shown in Figure 15, we used the state space tool to list all the state node information corresponding to the dead markings. The results show that state nodes 25, 28 and 29 are the cases where more than half of the task participants have chosen not to trust the model verifier v1. The smart contract removes the v1 verifier role from the list of legitimate verifiers. State node 32 indicates that only one task participant has chosen not to trust v1, and the corresponding information for v1 is updated to ("0x3001", 79). State node 33 is where all task participants choose to believe that there is no suspicious behaviour in v1, and this result is consistent with the simulated execution results in Table 5. In summary, the smart contract can complete trust arbitration as designed, only removing its verifier role authority if more than half of the task participants choose not to trust the model verifier. The fourth attribute is therefore guaranteed.



```
25:
Arbitration's_verifier 1: emp
Arbitration'Collaborators_v
1`("0x2002","model2")++
1`("0x2003","model3")
Arbitration'admin 1: 1`("0x
Arbitration'verifier 1: empty
Arbitration'confidence_vote
Arbitration'suspicious_beha
Arbitration'limit 1: empty
Arbitration'distrust 1: empt
Arbitration'trust 1: 1`0
Arbitration'initiate 1: empty
Arbitration'V_Users 1: 1`[]
```

```
28:
Arbitration's_verifier 1: emp
Arbitration'Collaborators_v
1`("0x2002","model2")++
1`("0x2003","model3")
Arbitration'admin 1: 1`("0x
Arbitration'verifier 1: empty
Arbitration'confidence_vote
Arbitration'suspicious_beha
Arbitration'limit 1: empty
Arbitration'distrust 1: empt
Arbitration'trust 1: 1`0
Arbitration'initiate 1: empty
Arbitration'V_Users 1: 1`[]
```

```
29:
Arbitration's_verifier 1: emp
Arbitration'Collaborators_v
1`("0x2002","model2")++
1`("0x2003","model3")
Arbitration'admin 1: 1`("0x
Arbitration'verifier 1: empty
Arbitration'confidence_vote
Arbitration'suspicious_beha
Arbitration'limit 1: empty
Arbitration'distrust 1: empty
Arbitration'trust 1: 1`1
Arbitration'initiate 1: empty
Arbitration'V_Users 1: 1`[]
```

```
32:
Arbitration's_verifier 1: empty
Arbitration'Collaborators_v 1: 1`("0x200
1`("0x2002","model2")++
1`("0x2003","model3")
Arbitration'admin 1: 1`("0x1001",1,1,"ad
Arbitration'verifier 1: empty
Arbitration'confidence_vote 1: 1`"susepc
Arbitration'suspicious_behavior 1: empty
Arbitration'limit 1: empty
Arbitration'distrust 1: 1`1
Arbitration'trust 1: empty
Arbitration'initiate 1: empty
Arbitration'V_Users 1: 1`[("0x3001",79)]
```

```
33:
Arbitration's_verifier 1: empty
Arbitration'Collaborators_v 1: 1`("0x200
1`("0x2002","model2")++
1`("0x2003","model3")
Arbitration'admin 1: 1`("0x1001",1,1,"ad
Arbitration'verifier 1: empty
Arbitration'confidence_vote 1: 1`"susepc
Arbitration'suspicious_behavior 1: empty
Arbitration'limit 1: empty
Arbitration'distrust 1: 1`0
Arbitration'trust 1: empty
Arbitration'initiate 1: empty
Arbitration'V_Users 1: 1`[("0x3001",80)]
```

Figure 15. The execution results of status formula and query statement of Arbitration layer.

5. Experiment and Case Study

In this section, we verify the usability of TDLearning. After the mechanism was deployed, model-fusion validation and collaboration cases were presented. The experimental results demonstrate that the mechanism can provide a trusted and fair collaborative infrastructure and effectively aggregate dispersed data resources to develop global models.

5.1. Platform

Firstly, we used Ganache-cli to simulate a remotely accessible private Ethereum blockchain. Secondly, the smart contract was developed by using the Solidity language and compiled and deployed by using the Truffle framework. Finally, we implement the interaction with the smart contract through the Node.js and Web3.js languages. The computational tasks in collaborative learning are performed in Python with the PyTorch deep learning framework, and pretrained models are stored by using IPFS.

5.2. Model-Fusion Validation

5.2.1. Experimental Scenario Setup

To aggregate the value of dispersed data resources, the collaborative-learning mechanism uses a model-fusion method based on feature fusion to achieve collaborative model training and proposes the use of an attention mechanism to precisely guide the model-fusion process. In order to verify the effectiveness of this mechanism for collaborative model training, a collaborative task based on the MNIST [35] dataset and the CIFAR10 [36] dataset is published by the task owner, and the global model classification accuracy is used as the evaluation index to analyse and evaluate the effectiveness of model fusion. The experiment set up one task owner and four task participants for a total of five collaboration nodes, with the collaboration goal of improving the global model accuracy. The structure of the pretrained model used by the collaborative nodes and the training parameters are shown in Table 6. Based on the differences in data distribution between collaborating nodes, the following three types of collaboration scenarios are experimentally set up to analyse the adaptability of the model-fusion approach to different data distributions.

Scenario A: The data distribution in this scenario is shown in Table 7, with each node's local data being independently and identically distributed.

Scenario B: The distribution of data in this scenario is shown in Table 8, where all four task participants had data samples from only some of the categories but the same amount of data from different labels.

Scenario C: In this scenario, the task owner has the same data as in the previous two scenarios, and the distribution of the task participants' data is shown in Figure 16. The horizontal axis in the figure represents the sample category, and the vertical axis represents the number of samples.

Table 6. Pretrained model structure and training parameters.

	CNN-MNIST	CNN-CIFAR10
Convolution layer 1	16 5 × 5 kernels	64 5 × 5 kernels
Pooling layer 1	2 × 2 max pooling	2 × 2 max pooling
Convolution layer 2	32 5 × 5 kernels	64 5 × 5 kernels
Pooling layer 2	2 × 2 max pooling	2 × 2 max pooling
Fully connected layer	128 units (with ReLU activation)	512 units (with ReLU activation)
Output layer	Softmax	Softmax
Optimiser	Adam	Adam
Learning rate	0.001	0.001
Batch size	64	64
Max local-training epochs	20	20
Max fine-tuning epochs	20	20

Table 7. Data distribution in scenario A (dataset: MNIST/CIFAR10).

Collaborative Nodes	*To*	*Tp* 1	*Tp* 2	*Tp* 3	*Tp* 4
Amount	1000/10,000	600/3000	600/3000	600/3000	600/3000
Label	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}				

Table 8. Data distribution in scenario B (dataset: MNIST/CIFAR10).

Collaborative Nodes	*To*	*Tp* 1	*Tp* 2	*Tp* 3	*Tp* 4
Amount	1000/10,000	600/3000	600/3000	600/3000	600/3000
Label	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	{0, 2, 3, 4, 5, 7}	{1, 2, 4, 7, 8, 9}	{1, 3, 5, 6, 7, 9}	{0, 3, 4, 5, 6, 8}

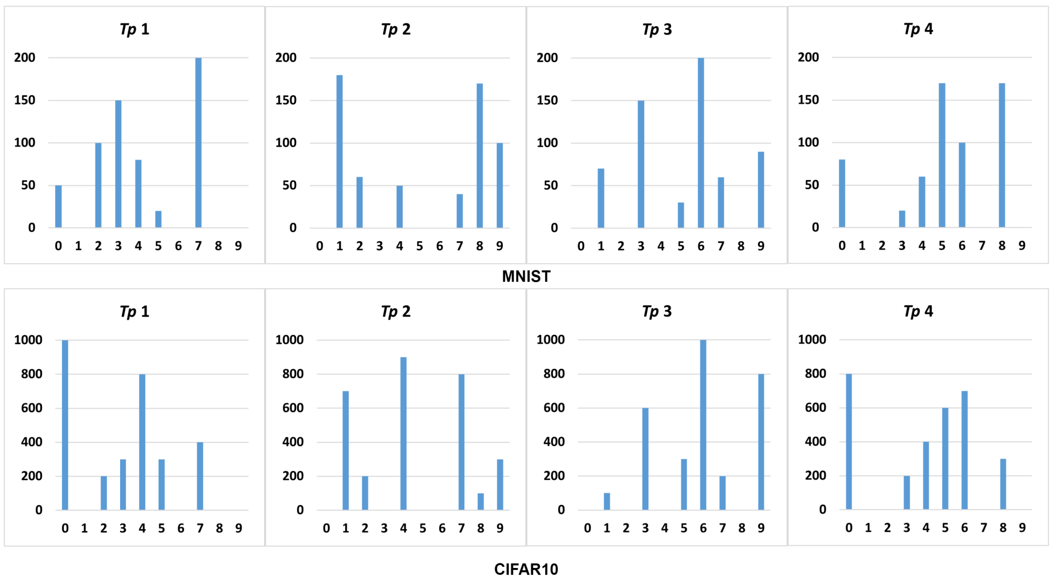


Figure 16. Data distribution in scenario C.

Based on the above experimental scenario, the method proposed in [26] (F-Concat) is compared with the feature-fusion method based on the attention mechanism proposed in this paper (F-Attention). Meanwhile, the Federated Average Algorithm (FedAvg) in Federated Learning is introduced as a benchmark, and the five collaborating nodes in

Federated Learning are trained with the following settings. In the MNIST dataset scenario, the number of local training rounds for the collaborative nodes is 20 and the maximum number of global model updates is 100, while in the CIFAR10 dataset scenario, the number of local training rounds for the collaborative nodes is 20 and the maximum number of global model updates is 50. In addition, the node-selection process is ignored in the global model update process, i.e., all nodes are selected to complete the global model update each time. For the processing of private data, the two steps are used. On the one hand, task participants register their local private data by using a data-management contract, which acts as a data node participating in collaborative-learning tasks. On the other hand, task participants use a specified model structure to develop pretrained models locally by using their private data.

5.2.2. Experimental Conclusions and Analysis

Based on the above experimental setup, the experimental results for the three types of collaboration scenarios are shown in Figure 17. The vertical axis in the figure indicates the average global model accuracy, and the horizontal axis indicates the number of fusion models. As shown in the experimental results, the classification accuracy of the global model developed by the proposed F-Attention improved steadily with the increase in the number of pretrained models fused in different collaboration scenarios, which effectively completed the fusion of the pretrained models. The F-Concat performed the worst, with degradation of the global model performance occurring in several scenarios, i.e., the global model accuracy decreased as the number of global-model-fusion pretrained models increased. The experimental results show that the attention mechanism used by the F-Attention achieves accurate guidance for feature fusion and avoids the degradation of the global model performance when a simple concatenation of features is performed. At the same time, the experimental results also indicate that the FedAvg has significant advantages in scenario one where the local data of collaborative nodes is independently and identically distributed. Its final global model classification accuracy is superior to the model-fusion method based on feature fusion. In scenarios two and three with heterogeneous data distribution, the F-Attention and the FedAvg method ended up with similar global model classification accuracies. The advantage of F-Attention is the efficiency of model fusion, which requires only one round of parameter interaction to complete the model fusion with a time complexity of $O(1)$, while FedAvg requires multiple rounds of parameter interaction with a time complexity of $O(n)$, where n denotes the number of rounds of parameter interaction. The effectiveness and usability of the methodology of the paper are verified by evaluating the model-fusion technique.

5.3. Case Study

In order to fully analyse and verify the usability of TDLearning, a typical case is implemented. A university released a collaborative task based on the F-MNIST [37] dataset, and four research institutions had the type of data and decided to participate in a collaborative effort to develop a global model together through collaborative model training. The university as the task owner has 2000 training samples containing 200 data samples for each of the 10 categories from 0 to 9 and has all 10,000 test samples. The four research institutions, as task participants, each have 2000 training samples, with heterogeneous data distribution due to different data sources. Each of the five collaborative nodes used the following pretrained model structure: a CNN with two 5×5 convolutional layers (the first with 16 channels while the second with 32, each followed by a ReLU activation and 2×2 max pooling), a fully connected layer with 256 units (with the ReLU activation function) and a final softmax output layer. The model training parameters were set as follows: learning rate: 0.001, batch size: 64, optimiser: Adam, max local training epochs: 30, max fine-tuning epochs: 20.

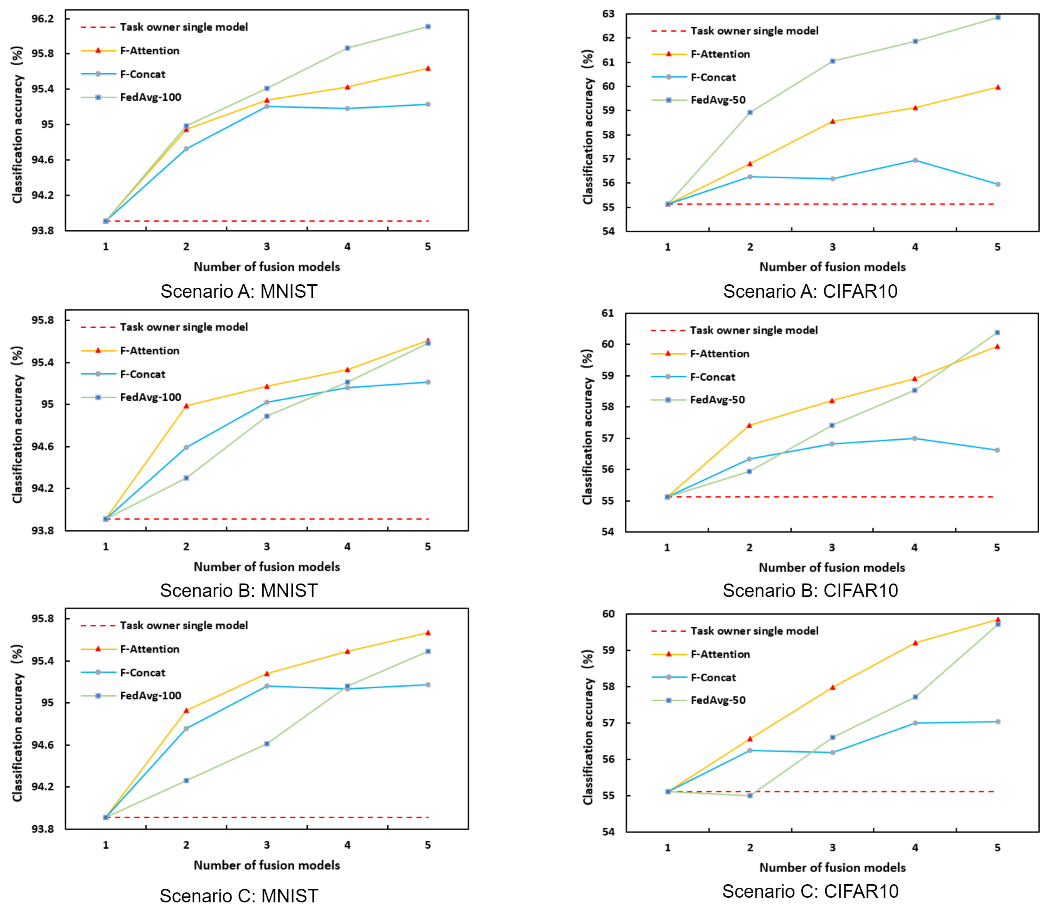
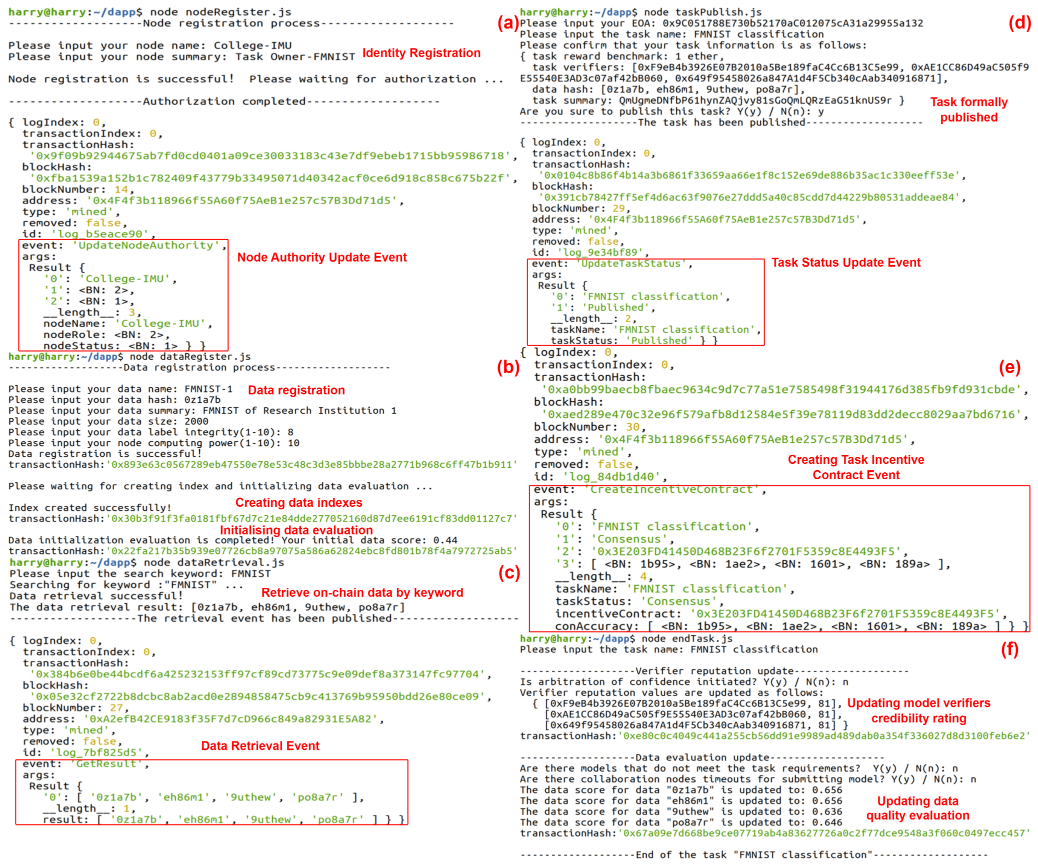


Figure 17. Experimental results in three scenarios: in different collaboration scenarios, with the increase in the number of fused pretrained models, the classification accuracy of the global model developed based on F-Attention has been steadily improved.

5.3.1. Task-Information-Initialisation Phase

Steps (a) to (c) in Figure 18 represent collaborative node-identity registration and authorisation, data registration and on-chain data retrieval, respectively. The prerequisite for collaborative nodes to participate in collaborative tasks is the registration of identity information and the assignment of role authority, i.e., their Ethereum account address is tied to the authority information. Step (a) shows the task owner node-identity registration and role-authority-assignment process, where the smart contract assigns the corresponding role authority to the node after its identity information has been successfully registered. As shown in this diagram, the assignment of authority triggers the UpdateNodeAuthority event, indicating that the node has completed the authorisation process, and the authorisation process is recorded and interacted with in an event manner. The identity-registration and authorisation process for the rest of the collaborating nodes is the same as that of the node, after which each node has the authority to participate in the on-chain collaboration.

Step (b) shows the data-registration process for a research institution, where the smart contract completes the data registration and returns the transaction record information. As shown in this step, once the basic data information is filled in, the data are successfully registered and added to the data list via the DMC. The smart contract then creates an inverted index for the data and initialises its data-quality evaluation. Subsequently, the remaining three research institutions all completed data registration through the above steps with data hashes of eh86m1, 9uthew and po8a7r, respectively. Step (c) shows the data-retrieval process; after successful retrieval, the smart contract returns the data hash value corresponding to the retrieval keyword and triggers the data-retrieval event.


(a)
Identity Registration
Node Authority Update Event
(b)
Data registration
Creating data indexes
Initialising data evaluation
(c)
Retrieve on-chain data by keyword
Data Retrieval Event
(d)
Task formally published
Task Status Update Event
(e)
Creating Task Incentive Contract Event
(f)
Updating model verifiers credibility rating
Updating data quality evaluation

```
harry@harry:~/dapp$ node nodeRegister.js
-------------------Node registration process-------------------

Please input your node name: College-IMU
Please input your node summary: Task Owner-FMNIST

Node registration is successful!  Please waiting for authorization ...

-------------------Authorization completed-------------------

{ logIndex: 0,
  transactionIndex: 0,
  transactionHash:
   '0x9f09b92944675ab7fd0cd0401a09ce30033183c43e7df9ebeb1715bb95986718',
  blockHash:
   '0xfba1539a152b1c782409f43779b33495071d40342acf0ce6d918c858c675b22f',
  blockNumber: 14,
  address: '0x4F4f3b118966f55A60f75AeB1e257c57B3Dd71d5',
  type: 'mined',
  removed: false,
  id: 'log_b5eace90',
  event: 'UpdateNodeAuthority',
  args:
   Result {
     '0': 'College-IMU',
     '1': <BN: 2>,
     '2': <BN: 1>,
     __length__: 3,
     nodeName: 'College-IMU',
     nodeRole: <BN: 2>,
     nodeStatus: <BN: 1> } }
harry@harry:~/dapp$ node dataRegister.js
-------------------Data registration process-------------------

Please input your data name: FMNIST-1
Please input your data hash: 0z1a7b
Please input your data summary: FMNIST of Research Institution 1
Please input your data size: 2000
Please input your data label integrity(1-10): 8
Please input your node computing power(1-10): 10
Data registration is successful!
transactionHash:'0x893e63c0567289eb47550e78e53c48c3d3e85bbbe28a2771b968c6ff47b1b911'

Please waiting for creating index and initializing data evaluation ...

Index created successfully!
transactionHash:'0x30b3f91f3fa0181fbf67d7c21e84dde277052160d87d7ee6191cf83dd01127c7'

Data initialization evaluation is completed! Your initial data score: 0.44
transactionHash:'0x22fa217b35b939e07726cb8a97075a586a62824ebc8fd801b78f4a7972725ab5'
harry@harry:~/dapp$ node dataRetrieval.js
Please input the search keyword: FMNIST
Searching for keyword :"FMNIST" ...
Data retrieval successful!
The data retrieval result: [0z1a7b, eh86m1, 9uthew, po8a7r]
-------------------The retrieval event has been published-------------------

{ logIndex: 0,
  transactionIndex: 0,
  transactionHash:
   '0x384b6e0be44bcdf6a425232153ff97cf89cd73775c9e09def8a373147fc97704',
  blockHash:
   '0x05e32cf2722b8dcbc8ab2acd0e2894858475cb9c413769b95950bdd26e80ce09',
  blockNumber: 27,
  address: '0xA2efB42CE9183f35F7d7cD966c849a82931E5A82',
  type: 'mined',
  removed: false,
  id: 'log_7bf825d5',
  event: 'GetResult',
  args:
   Result {
     '0': [ '0z1a7b', 'eh86m1', '9uthew', 'po8a7r' ],
     __length__: 1,
     result: [ '0z1a7b', 'eh86m1', '9uthew', 'po8a7r' ] } }
```

```
harry@harry:~/dapp$ node taskPublish.js
Please input your EOA: 0x9C051788E730b52170aC012075cA31a29955a132
Please input the task name: FMNIST classification
Please confirm that your task information is as follows:
{ task reward benchmark: 1 ether,
  task verifiers: [0xF9eB4b3926E07B2010a5Be189faC4Cc6B13C5e99, 0xAE1CC86D49aC505f9
E55540E3AD3c07af42bB060, 0x649f95458026a847A1d4F5Cb340cAab340916871],
  data hash: [0z1a7b, eh86m1, 9uthew, po8a7r],
  task summary: QmUgmeDNfbP61hynZAQjvy81sGoQmLQRzEaG51knUS9r }
Are you sure to publish this task? Y(y) / N(n): y
-------------------The task has been published-------------------

{ logIndex: 0,
  transactionIndex: 0,
  transactionHash:
   '0x0104c8b86f4b14a3b6861f33659aa66e1f8c152e69de886b35ac1c330eeff53e',
  blockHash:
   '0x391cb78427ff5ef4d6ac63f9076e27ddd5a40c85cdd7d44229b80531addeae84',
  blockNumber: 29,
  address: '0x4F4f3b118966f55A60f75AeB1e257c57B3Dd71d5',
  type: 'mined',
  removed: false,
  id: 'log_9e34bf89',
  event: 'UpdateTaskStatus',
  args:
   Result {
     '0': 'FMNIST classification',
     '1': 'Published',
     __length__: 2,
     taskName: 'FMNIST classification',
     taskStatus: 'Published' } }
{ logIndex: 0,
  transactionIndex: 0,
  transactionHash:
   '0xa0bb99baecb8fbaec9634c9d7c77a51e7585498f31944176d385fb9fd931cbde',
  blockHash:
   '0xaed289e470c32e96f579afb8d12584e5f39e78119d83dd2decc8029aa7bd6716',
  blockNumber: 30,
  address: '0x4F4f3b118966f55A60f75AeB1e257c57B3Dd71d5',
  type: 'mined',
  removed: false,
  id: 'log_84db1d40',
  event: 'CreateIncentiveContract',
  args:
   Result {
     '0': 'FMNIST classification',
     '1': 'Consensus',
     '2': '0x3E203FD41450D468B23F6f2701F5359c8E4493F5',
     '3': [ <BN: 1b95>, <BN: 1ae2>, <BN: 1601>, <BN: 189a> ],
     __length__: 4,
     taskName: 'FMNIST classification',
     taskStatus: 'Consensus',
     incentiveContract: '0x3E203FD41450D468B23F6f2701F5359c8E4493F5',
     conAccuracy: [ <BN: 1b95>, <BN: 1ae2>, <BN: 1601>, <BN: 189a> ] } }
harry@harry:~/dapp$ node endTask.js
Please input the task name: FMNIST classification

-------------------Verifier reputation update-------------------
Is arbitration of confidence initiated? Y(y) / N(n): n
Verifier reputation values are updated as follows:
  { [0xF9eB4b3926E07B2010a5Be189faC4Cc6B13C5e99, 81],
    [0xAE1CC86D49aC505f9E55540E3AD3c07af42bB060, 81],
    [0x649f95458026a847A1d4F5Cb340cAab340916871, 81] }
transactionHash:'0xe80c0c4049c441a255cb56dd91e9989ad489dab0a354f336027d8d3100feb6e2'

-------------------Data evaluation update-------------------
Are there models that do not meet the task requirements?  Y(y) / N(n): n
Are there collaboration nodes timeouts for submitting model? Y(y) / N(n): n
The data score for data "0z1a7b" is updated to: 0.656
The data score for data "eh86m1" is updated to: 0.656
The data score for data "9uthew" is updated to: 0.636
The data score for data "po8a7r" is updated to: 0.646
transactionHash:'0x67a09e7d668be9ce07719ab4a83627726a0c2f77dce9548a3f060c0497ecc457'

-------------------End of the task "FMNIST classification"-------------------
```

Figure 18. (**a**–**c**): task-information-initialisation phase, (**d**–**f**): task-execution phase.

In the information-initialisation phase of the collaborative task, each collaborative node completed the registration of identity information and was assigned the corresponding role authority by the smart contract. The four research institutes then uploaded local data information to the blockchain as participants of the task, and the smart contract completed data indexing and a data-quality evaluation, establishing a trusted on-chain database for the collaborative task. In summary, the allocation of role authority and the establishment of the trusted database marked the completion of the initialisation of the collaborative task information.

5.3.2. Task-Execution Phase

After identifying the task participants and model verifiers involved in the collaboration, the task owner invokes the task initialisation function of the task-management contract to register the collaborative task with the blockchain and at the same time pledge the task incentive to the contract. As in step (d), when the smart contract determines that sufficient task incentives have been pledged, the task owner can officially release the task, and the task status is updated to formal publication. After the collaborative task is officially released, the task owner first sends the standard test set of the pretrained model to all model verifiers in an on-chain encrypted manner. The task participants then send the pretrained models to all model verifiers in an on-chain encrypted manner after they have been developed, and the model verifiers evaluate the pretrained models with the standard test sets and upload the model-verification results in an encrypted manner via the TMC. When the TMC determines that a consensus on model validation has been reached, it then creates a task-incentive contract for the task and publishes a corresponding event to incentivise task participants and model verifiers with a preset incentive benchmark and model-validation results as indicators, as shown in step (e).

After the task owner received the verified pretrained models, multiple models were fused by using a model-fusion method based on feature fusion to develop a global model.

Figure 19 shows the classification accuracy of the multiple pretrained models as well as the global model, showing that the average classification accuracy of the global model steadily increases as the number of fused models increases. At the end of the task incentive, the supervisor invokes the authority-management contract and the data-evaluation contract, respectively, to update the model verifier credibility value and the collaborative data-quality evaluation associated with this collaborative task, as shown in step (f). Once the above steps have been completed, the task is set to finished and this collaborative task is over.

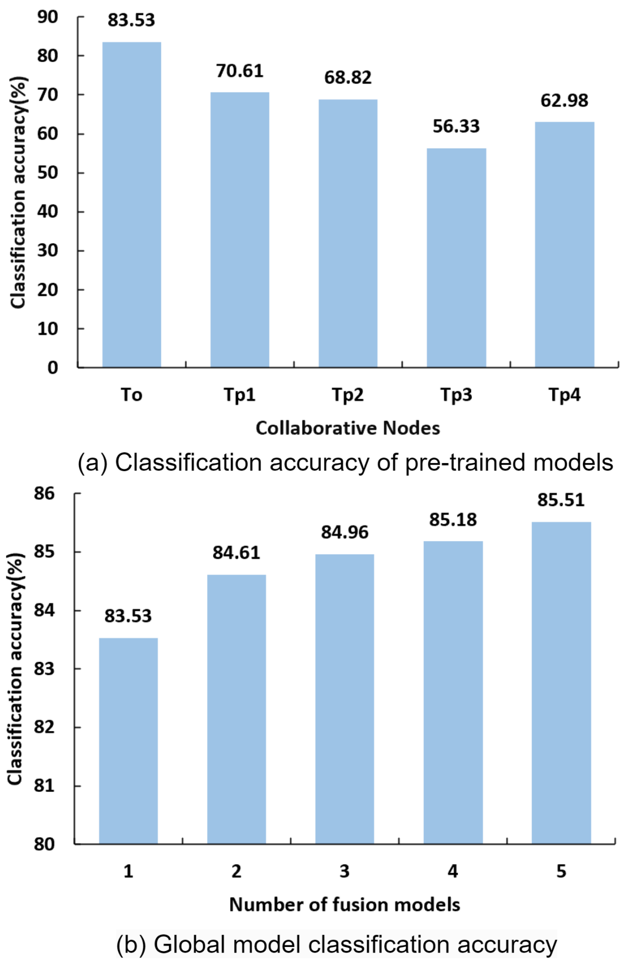


Figure 19. Pretrained model and global model classification accuracy.

5.3.3. Trustworthiness and Fairness

Trustworthiness: Firstly, the smart-contract-based authority-management function implements the access mechanism for distributed collaboration and completes the role-based authority assignment. The binding of identity information and authority encourages collaborative nodes to act honestly and avoid the influence of malicious nodes on collaborative tasks. Secondly, the blockchain records the collaborative tasks and the collaborative interactions between nodes, and its traceable and tamper-evident storage content strongly restrains the possible malicious behaviour of nodes. Thirdly, smart contracts control the process of collaborative tasks and complete the automatic incentive for collaborative nodes, freeing them from reliance on trusted third parties. Finally, the introduction of events in the mechanism further increases the trustworthiness of the collaboration mechanism. On the one hand, events serve as signals for smart contracts to respond to and trigger collaboration behaviours, describing the blockchain transaction behaviour at a specific or specified moment, and collaborating nodes can listen to the events to obtain the collaboration process and execute the corresponding collaboration behaviours. On the other hand, the Ethereum platform records events in the form of logs, and collaborative nodes can retrieve the history of collaborative actions through the logs.

Fairness: The collaborative-learning mechanism introduces a model-verification consensus mechanism to evaluate the contribution of different task participants to the collaborative task. Multiple model verifiers independently verify the pretrained models submitted by task participants and reach a consensus on the verification through a smart contract and finally incentivise the task participants with the model accuracy. The model-verification consensus and task incentive are both accomplished by the smart contract, which protects the interests of both the task owner and task participants and encourages more data nodes to participate in the collaborative task.

6. Conclusions

To address the problems of the ineffective aggregation of dispersed data resources and the low trustworthiness of existing distributed-collaborative-learning architectures, we propose TDLearning. The mechanism uses blockchain as the distributed collaborative environment, and smart contracts define and encapsulate distributed collaborative behaviours, relationships and specifications. At the same time, based on the smart contract collaboration architecture and transfer learning, a model-fusion method based on feature fusion is proposed to realise distributed-model collaborative training and aggregate the value of dispersed data resources to improve the model performance. We analyse the trustworthiness of smart contracts by means of formal methods and analyse the usability of the proposed mechanism by means of experiments and a case study. The experimental results show that the mechanism provides a credible and fair collaboration infrastructure and organises decentralised data resources to accomplish the collaborative training of models to develop effective global models. In subsequent studies, we would combine and compare with other blockchain technologies to further improve the effectiveness of distributed collaborative learning. Meanwhile, the application scenarios of the model would be extended to verify the practicality of the architecture.

Author Contributions: Conceptualisation, J.L., X.H. and K.L.; formal analysis, J.L. and X.H.; methodology, J.L. and X.H.; supervision, K.L.; validation, X.H.; writing—original draft, X.H.; writing—review and editing, J.L. and K.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Natural Science Foundation of Inner Mongolia of China (No. 2023ZD18); the Inner Mongolia Science and Technology Plan Project (No. 2020GG0187); and the Engineering Research Center of Ecological Big Data, Ministry of Education.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Roh, Y.; Heo, G.; Whang, S.E. A Survey on Data Collection for Machine Learning: A Big Data—AI Integration Perspective. *IEEE Trans. Knowl. Data Eng.* **2019**, *33*, 1328–1347. [CrossRef]
2. Issa, W.; Moustafa, N.; Turnbull, B.; Sohrabi, N.; Tari, Z. Blockchain-based federated learning for securing internet of things: A comprehensive survey. *ACM Comput. Surv.* **2023**, *55*, 1–43. [CrossRef]
3. Qammar, A.; Karim, A.; Ning, H.; Ding, J. Securing federated learning with blockchain: A systematic literature review. *Artif. Intell. Rev.* **2023**, *56*, 3951–3985. [CrossRef] [PubMed]
4. Yang, F.; Abedin, M.Z.; Hajek, P. An explainable federated learning and blockchain-based secure credit modeling method. *Eur. J. Oper. Res.* 2023, *in press*. [CrossRef]
5. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A.y. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
6. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated learning: Challenges, methods, and future directions. *IEEE Signal Process. Mag.* **2020**, *37*, 50–60. [CrossRef]
7. Toyoda, K.; Zhang, A.N. Mechanism design for an incentive-aware blockchain-enabled federated learning platform. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 395–403.

8. Lyu, L.; Yu, H.; Yang, Q. Threats to federated learning: A survey. *arXiv* **2020**, arXiv:2003.02133.
9. Guo, H.; Yu, X. A Survey on Blockchain Technology and its security. *Blockchain Res. Appl.* **2022**, *3*, 100067. [CrossRef]
10. Lin, S.; Zhang, L.; Li, J.; Ji, L.; Sun, Y. A survey of application research based on blockchain smart contract. *Wirel. Netw.* **2022**, *28*, 635–690. [CrossRef]
11. Wang, X.; Ren, X.; Qiu, C.; Xiong, Z.; Yao, H.; Leung, V.C.M. Integrating edge intelligence and blockchain: What, why, and how. *IEEE Commun. Surv. Tutorials* **2022**, *24*, 2193–2229. [CrossRef]
12. Khan, A.A.; Laghari, A.A.; Rashid, M.; Li, H.; Javed, A.R.; Gadekallu, T.R. Artificial intelligence and blockchain technology for secure smart grid and power distribution Automation: A State-of-the-Art Review. *Sustain. Energy Technol. Assess.* **2023**, *57*, 103282.
13. Shukla, A.; Lodha, N. Investigating the Role of Artificial Intelligence in Building Smart Contact on Blockchain. In Proceedings of the 2022 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 9–11 March 2022; pp. 1–6.
14. Kim, H.; Park, J.; Bennis, M.; Kim, S.-L. Blockchained on-device federated learning. *IEEE Commun. Lett.* **2019**, *24*, 1279–1283. [CrossRef]
15. Qu, Y.; Gao, L.; Luan, T.H.; Xiang, Y.; Yu, S.; Li, B.; Zheng, G. Decentralized privacy using blockchain-enabled federated learning in fog computing. *IEEE Internet Things J.* **2020**, *7*, 5171–5183. [CrossRef]
16. Wang, Y.; Su, Z.; Zhang, N.; Benslimane, A. Learning in the air: Secure federated learning for UAV-assisted crowdsensing. *IEEE Trans. Netw. Sci. Eng.* **2020**, *8*, 1055–1069. [CrossRef]
17. Lu, Y.; Huang, X.; Dai, Y.; Maharjan, S.; Zhang, Y. Blockchain and federated learning for privacy-preserved data sharing in industrial IoT. *IEEE Trans. Ind. Inform.* **2019**, *16*, 4177–4186. [CrossRef]
18. Harris, J.D.; Waggoner, B. Decentralized and collaborative AI on blockchain. In Proceedings of the IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 368–375.
19. Lugan, S.; Desbordes, P.; Brion, E.; Tormo, L.X.R.; Legay, A.; Macq, B. Secure architectures implementing trusted coalitions for blockchained distributed learning (TCLearn). *IEEE Access* **2019**, *7*, 181789–181799. [CrossRef]
20. Awan, S.; Li, F.; Luo, B.; Liu, M. Poster: A reliable and accountable privacy-preserving federated learning framework using the blockchain. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 2561–2563.
21. Miao, Y.; Liu, Z.; Li, H.; Choo, K.R.; Deng, R.H. Privacy-preserving Byzantine-robust federated learning via blockchain systems. *IEEE Trans. Inf. Forensics Secur.* **2022**, *17*, 2848–2861. [CrossRef]
22. Ma, C.; Li, J.; Shi, L.; Ding, M.; Wang, T.; Han, Z.; Poor, H.V. When federated learning meets blockchain: A new distributed learning paradigm. *IEEE Comput. Intell. Mag.* **2022**, *17*, 26–33. [CrossRef]
23. Bozkurt, A.; Ucar, H. Blockchain technology as a bridging infrastructure among formal, non-formal, and informal learning processes. In *Research Anthology on Adult Education and the Development of Lifelong Learners*; Information Science Reference: Hershey, PA, USA, 2021; pp. 959–970.
24. Alsobeh, A.; Woodward, B. AI as a Partner in Learning: A Novel Student-in-the-Loop Framework for Enhanced Student Engagement and Outcomes in Higher Education. In Proceedings of the 24th Annual Conference on Information Technology Education, Marietta, GA, USA, 11–14 October 2023; pp. 171–172.
25. Ramanan, P.; Nakayama, K. Baffle: Blockchain based aggregator free federated learning. In Proceedings of the 2020 IEEE International Conference on Blockchain (Blockchain), Rhodes Island, Greece, 2–6 November 2020; pp. 72–81.
26. Mendis, G.J.; Wu, Y.; Wei, J.; Sabounchi, M.; Roche, R. A blockchain-powered decentralized and secure computing paradigm. *IEEE Trans. Emerg. Top. Comput.* **2020**, *9*, 2201–2222. [CrossRef]
27. Ouyang, L.; Yuan, Y.; Wang, F.-Y. Learning markets: An AI collaboration framework based on blockchain and smart contracts. *IEEE Internet Things J.* **2020**, *9*, 4273–14286. [CrossRef]
28. Ouyang, L.; Yuan, Y.; Cao, Y.; Wang, F.-Y. A novel framework of collaborative early warning for COVID-19 based on blockchain and smart contracts. *Inf. Sci.* **2021**, *570*, 124–143. [CrossRef]
29. Oktian, Y.E.; Stanley, B.; Lee, S.-G. Building Trusted Federated Learning on Blockchain. *Symmetry* **2022**, *14*, 1407. [CrossRef]
30. Benet, J. Ipfs-content addressed, versioned, p2p file system. *arXiv* **2014**, arXiv:1407.3561.
31. Neyshabur, B.; Sedghi, H.; Zhang, C. What is being transferred in transfer learning? *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 512–523.
32. Jensen, K.; Kristensen, L.M. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*; Springer Science & Business Media: Berlin, Germany, 2009.
33. Jensen, K.; Kristensen, L.M.; Wells, L. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *Int. J. Softw. Tools Technol. Transf.* **2007**, *9*, 213–254. [CrossRef]
34. Cheng, A.; Christensen, S.; Mortensen, K.H. *Model Checking Coloured Petri Nets-Exploiting Strongly Connected Components*; DAIMI Report Series; The Royal Danish Library: Copenhagen, Denmark, 1997; Volume 519, pp. 1–17.
35. LeCun, Y. The MNIST Database of Handwritten Digits. 1998. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 20 November 2023).

36. Krizhevsky, A. Learning multiple layers of features from tiny images. In *Handbook of Systemic Autoimmune Diseases*; University of Toronto: Toronto, ON, Canada, 2009.
37. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017**, arXiv:1708.07747.