electronics

MDPI

Article

TrustDFL: A Blockchain-Based Verifiable and Trusty Decentralized Federated Learning Framework

Jinsheng Yang [1], Wenfeng Zhang [1], Zhaohui Guo [1] and Zhen Gao [2,*]

[1] School of Microelectronics, Tianjin University, Tianjin 300072, China; jsyang@tju.edu.cn (J.Y.); 2020232164@tju.edu.cn (W.Z.); 2017232048@tju.edu.cn (Z.G.)
[2] School of Electrical Automation and Information Engineering, Tianjin University, Tianjin 300072, China
* Correspondence: zgao@tju.edu.cn

Abstract: Federated learning is a privacy-preserving machine learning framework where multiple data owners collaborate to train a global model under the orchestra of a central server. The local training results from trainers should be submitted to the central server for model aggregation and update. Busy central server and malicious trainers can introduce the issues of a single point of failure and model poisoning attacks. To address the above issues, the trusty decentralized federated learning(called TrustDFL) framework has been proposed in this paper based on the zero-knowledge proof scheme, blockchain, and smart contracts, which provides enhanced security and higher efficiency for model aggregation. Specifically, Groth 16 is applied to generate the proof for the local model training, including the forward and backward propagation processes. The proofs are attached as the payloads to the transactions, which are broadcast into the blockchain network and executed by the miners. With the support of smart contracts, the contributions of the trainers could be verified automatically under the economic incentive, where the blockchain records all exchanged data as the trust anchor in multi-party scenarios. In addition, IPFS (InterPlanetary File System) is introduced to alleviate the storage and communication overhead brought by local and global models. The theoretical analysis and estimation results show that the TrustDFL efficiently avoids model poisoning attacks without leaking the local secrets, ensuring the global model's accuracy to be trained.

Keywords: decentralized federated learning; blockchain; verifiability; zero-knowledge proof (ZKP); zk-SNARK

Citation: Yang, J.; Zhang, W.; Guo, Z.; Gao, Z. TrustDFL: A Blockchain-Based Verifiable and Trusty Decentralized Federated Learning Framework. *Electronics* **2024**, *13*, 86. https://doi.org/10.3390/electronics13010086

Academic Editor: Mehdi Sookhak

Received: 27 November 2023
Revised: 19 December 2023
Accepted: 22 December 2023
Published: 24 December 2023

1. Introduction

As a branch of artificial intelligence, machine learning (ML) is a statistical model driven by big data to tackle complex problems, and has been used widely in various fields, such as image recognition, image segmentation, and natural language processing [1–3]. With the development of the deep neural network (DNN), training a larger model requires more data and computing power, which usually is delegated to the centralized data center or cloud providers [4]. For example, the machine learning as a service (MLaaS) framework provides prediction results based on the trained models maintained on the centralized server, which needs to collect the training data set from the edge devices in advance and causes the risk of privacy leakage [5]. Federated learning (FL) is proposed for multiple participants to train a global model corroboratively without exposure of the local data [6,7]. As shown in Figure 1, the FL framework is formed by edge devices acting as trainers and the server acting as the aggregator. The global model training in FL runs multiple rounds. For each round, there are four steps, as follows [8]: (1) global model distribution: the aggregator distributes the global model to the selected trainer; (2) local training: relying on the local training data set, the selected trainers update the parameters based on the published global model; (3) update upload: the selected trainers upload the trained results (the updated weights or gradients), instated of the raw training data, to the server, which avoids privacy

leakage; (4) model aggregation: the server aggregates the updates and generates the new global model for the next round. The target global model to be trained in FL could have any architectures, such as multi-layer perceptron (MLP) and convolutional neural networks (CNN). The standard implementation of FL is FederatedAveraging (FedAvg), where the updates for the global model could be aggregated through a weighted average [9].

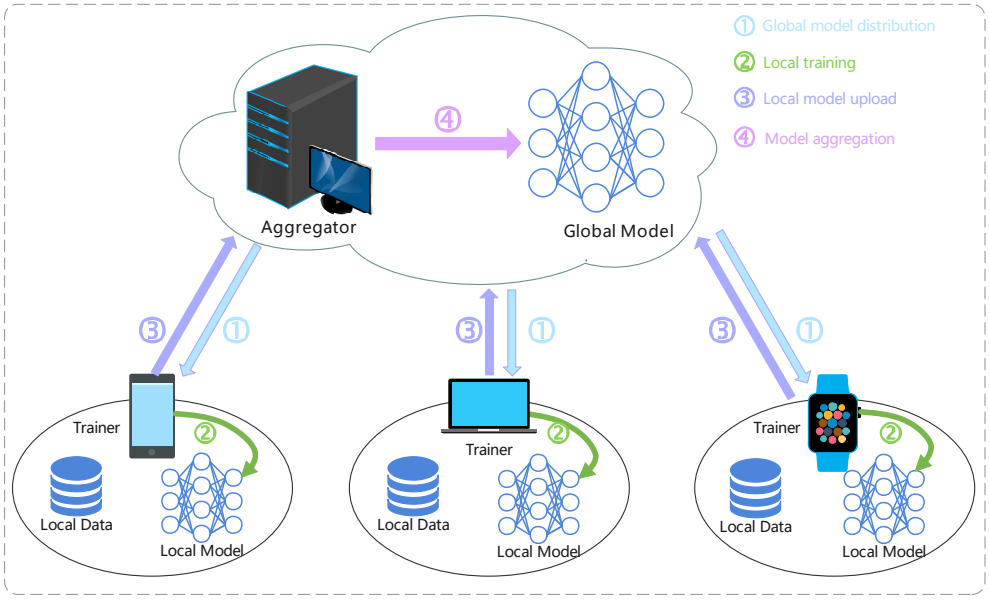

Figure 1. FL framework and global model training process.

However, there are two major issues in the FL framework. First, it is hard for the aggregators to check the validity of the updates uploaded by the trainers, which brings the risk of model poisoning attacks and reduces model availability [10–12]. The malicious trainers could launch model poisoning attacks by uploading invalid updates (for example, random float numbers generated randomly), which could interfere with the aggregation or convergence process, and reduce the global model accuracy. Since the raw training data on each trainer cannot be revealed to the aggregator for privacy-preserving reasons, there is no efficient method to check the contributions of the trainers. There are some robustness aggregation methods, such as Krum and Trim-means, which remove the updates dissimilar from others based on the statistical analysis before the aggregation process [13,14]. On one hand, these statistical methods are imprecise because trainers have different data distributions. On the other hand, by the robustness aggregation, all workloads for verification are on the aggregator sides, which causes a computation bottleneck.

Second, in the traditional FL framework (known as centralized FL, CFL), global model training is orchestrated by the centralized server, which faces a single point of failure problem and cannot meet the requirement for decentralization in the cross-silo scenarios. The CFL could be transformed into a decentralized FL (DFL) by replacing the designated server with a cluster of aggregators from different organizations [15,16]. There are two issues in the current DFL framework, as follows: (1) like the trainers, the contributions of the aggregators should also be verified efficiently. (2) All contributions should be recorded honestly to construct trust, and the incentive mechanism is needed to encourage all participants to adhere to the protocol honestly.

Various schemes were proposed to address the issues above, and most introduce blockchain for decentralized FL. Under the framework of establishing decentralized federated learning, the authors in [17] proposed a new committee consensus mechanism to complete the tasks of gradient selection and block generation. The elected committee serves as the miner's responsibility and uses K-fold cross-validation to verify and score the trainer's updates. The authors in [18] elected a committee to judge the reliability of the model parameters by verifying whether the trainer's training time is proportional to

the data size. There are also some solutions that integrate blockchain, smart contracts, and the zero-knowledge proof (ZKP) system. The blockchain is adopted as the trust anchor, the ZKP is used for verifying contributions, and the smart contracts are used for the automatic execution of contribution verification and reward distribution. Different from the robustness aggregation and cross-validation, the verification methods based on the ZKP enable the trainers to convince the aggregators that the updates are valid by providing proof indicating the correct execution of the defined computation. For example, [19–21] adopted the zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) to prove the correctness of the prediction results without revealing the model parameters in the MlaaS scenarios, where only the forward prorogation (FP) process and two-party scenarios are considered. As proposed in [22,23], the schemes integrate the ZKPs and blockchain to construct the verifiable DFL framework, where the verification for updates is viewed as part of the consensus process, and miners are responsible for the verification before recording the updates into the blocks. In [24], ZKP-based verification is executed by smart contracts. However, these schemes use the blockchain as the black box or the third-party platform. The global model training of the FL is not integrated into the transaction lifecycle of the blockchain.

This paper proposes a trustworthy and verifiable decentralized federated learning framework (TrustDFL) based on blockchain and ZKP. The issues of a single point of failure problem and model poisoning attacks could be addressed efficiently by the proposed TrustDFL framework with limited computational overhead for on-chain verification. Moreover, the storage burden of the blockchain could be alleviated by dumping global and local models into the IPFS. The contributions of this paper are as follows:

1. We propose the TrustDFL framework, which integrates the global model training in the FL into the consensus process in the blockchain. The framework adopts zero-knowledge proof to establish the proof of the computation correctness of the local training process and the model aggregation process, which are implemented by smart contracts. Only the successfully verified models will be recorded on the blockchain.
2. We specifically use zk-SNARK to construct the proof system, which has the advantages of succinctness (small proof size) and efficiency (fast proof verification). We provide the construction detail of the proof for the correctness of local training (including the forward and back propagation processes). The proof for the aggregation process could be generated similarly based on the defined aggregation algorithm.
3. Considering the blockchain storage pressure, we introduce an IPFS to store the hash related to the model on the blockchain instead of the raw model parameters, which effectively avoids storage scalability issues for the blockchain.

TrustDFL can be applied to many scenarios. For example, in the current model of cloud computing, MLaaS, it is necessary to ensure the availability of the global model. In this case, the operator provides predictable service for pay using the trained global model, which is obtained by the DFL with data collected from sensors owned by users with different interests. TrustDFL can also be applied to the healthcare industry, where smart health applications utilize global models jointly trained by users with data to provide health monitoring services. Since incorrect predictions of health status can bring serious consequences, and personal health data is highly confidential, TrustDFL can ensure the reliability of the global model without leaking secrets, which is very suitable for this scenario [25].

The rest of this paper is organized as follows. Section 2 describes the basic technique. Section 3 introduces the detailed design of the TrustDFL. Theoretical analysis and simulation results for the TrustDFL are given in Sections 4 and 5, respectively. The paper discusses and concludes in Sections 6 and 7.

2. Background

This section introduces the basic techniques for constructing the TrustDFL, including the architecture of the multilayer perceptron (MLP), blockchain, and zero-knowledge proof (ZKP) system, in Sections 2.1–2.3, respectively.

2.1. Multi-Layer Perceptron (MLP)

In this paper, a multi-layer perceptron (MLP) is used as a case study to demonstrate the principle of TrustDFL. A multi-layer perceptron (MLP) is a fully connected feedforward neural network, typically including an input layer, a hidden layer, and an output layer, where all layers are formed by multiple neurons with weights and a bias vector to be trained [26]. Unlike the linear perceptron, the hidden and output layers are followed by the activation function f. In the MLP, there are two most commonly used activation functions: ReLU and sigmoid, shown as Equation (1) and Equation (2), respectively.

$$f(x) = max(0, x) \tag{1}$$

$$f(x) = \frac{1}{1 - e^{-x}} \tag{2}$$

The MLP can provide the prediction results based on the inputs known as forward propagation (FP) [27,28]. The weights and bias vectors could be trained by the backpropagation (BP). Typically, the training process in MLP includes serval epochs. For each epoch, the weights and biases are updated using the gradient descent algorithm. During the FP, the calculation of the i-th layer could be described as follows:

$$Y_i = f(X_i \bullet W_i + \mathbf{b}_i), \tag{3}$$

where X_i, Y_i, W_i, and $\mathbf{b}_i$ are the input matrix, output matrix, weight matrix, and bias vector of the i-th layer. The output of the i-th layer is the input of the $i+1$-th layer. $f(x)$ is the activation function.

In the output layer, a loss function L is used to estimate the gaps between the prediction results and the labels. Typically, for multi-category classification tasks, the cross-entropy is used as the loss function:

$$L = -\frac{1}{n}\sum_{j}\sum_{c=1}^{M} y_{jc} \bullet log(p_{jc}) \tag{4}$$

where n and M are the numbers of the samples and classes, respectively. The y_{jc} is the sign function, which is 1 when the j-th sample belongs to the c-th class ($y_{jc} = 0$, otherwise). p_{jc} is the probability that the j-th sample belongs to the c-th class. The target of the training process is minimizing the loss function by updating the weight matrices and bias vectors of all layers in the next epoch, which is realized by the gradient descent algorithm as follows:

$$\begin{cases} W_i' = W_i - \eta \nabla W_i \\ \mathbf{b}_i' = \mathbf{b}_i - \eta \nabla \mathbf{b}_i \end{cases} \tag{5}$$

where η is the learning rate. The ∇W_i and $\nabla \mathbf{b}_i$ are the gradients of the loss function regarding the weights and bias, which could be calculated based on the chain rules as:

$$\begin{cases} \nabla W_i = \frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial Y_i}\frac{\partial Y_i}{\partial F_i}\frac{\partial F_i}{\partial W_i} \\ \nabla \mathbf{b}_i = \frac{\partial L}{\partial \mathbf{b}_i} = \frac{\partial L}{\partial Y_i}\frac{\partial Y_i}{\partial F_i}\frac{\partial F_i}{\partial \mathbf{b}_i} \end{cases} \tag{6}$$

where $F_i = X_i \bullet W_i + \mathbf{b}_i$. The process of updating weights and biases according to their gradients is known as backpropagation, which will be repeated for multiple epochs until the model coverages.

MLP is an early and widely used machine learning model. It is often used for text classification, audio processing, image recognition, and some traditional machine learning tasks, such as classification, regression, clustering, etc. For large-scale data sets and image processing tasks, CNN is more suitable.

2.2. Blockchain and IPFS

Blockchain is the emerging distributed database system and computation paradigm underlying cryptocurrencies, such as BTC and ETH [29,30]. As the world computer and the finite state machine, the transaction is the basic process unit to drive the state translation. All transactions are maintained by the ledger formed by the blocks chained in chronological order, where the integrity of the records is guaranteed by the cryptographic techniques (such as hash and digital signature algorithms) and the consensus processes (such as proof-of-work and proof-of-stake). With the support of peer-to-peer, all blockchain nodes share the same data access rights, and only the data passing the public verification could be appended [31].

The smart contract is the key technology for the blockchain to develop decentralization applications (Dapp), build autonomous communities, and even construct smart cities [32,33]. Taking Ethereum as an example, the smart contract is the protocol that all participants should adhere to, which is implemented as the bytecodes running in the specific virtual machine, such as the Ethereum virtual machine (EVM). The smart contract could be triggered by the preset conditions and executed automatically, which excludes the reliance on the trusted third party and the human factor. Thus, the smart contract enables behavior customization for all nodes and public verification for all data, which empowers the blockchain to act as the trust anchor in multi-party cooperative scenarios. Most existing blockchains implement smart contracts and the corresponding Turing-complete programming language, such as the smart contract written by Solidity in Ethereum and the chain code written by Golang in the Hyperledger Fabric [34,35].

In Ethereum, there are two types of transactions: the ones for token transfer and the ones for contract deployment and calling [33,34]. For the former, the field of data is defaulted to empty. For the latter, the field of data is the contract bytecodes (for deployment) or the parameters (for the calling). All transactions follow the same lifecycle from issuance to confirmation as follows: (1) the node issues a new transaction and broadcasts it to the network; (2) the nodes receiving the new transaction check its validity and cache the valid transition into the local pool; (3) the nodes pack the transaction into a new block and compete for the authority block proposal through the consensus process; (4) the nodes receiving the new block validate the consensus result and the validity of all transaction, then append the block passing all the checks into the tip of the local chain.

Currently, the blockchain faces scalability issues, especially in terms of storage scalability. Since all nodes should maintain the ledger, there will be a huge storage burden for the resource-limited nodes with increasing block height. The storage scalability issue of the blockchain could be alleviated by dumping the ledger into the off-chain, such as the interplanetary file system (IPFS). The IPFS is the file system based on the distributed hash table, where the files are indexed by their unique IPFS hash values instead of the URLs (known as content-based addressing) [36]. As proposed in the paper by [37], by integrating the IPFS with the blockchain, the original ledger could be dumped into the IPFS, with only the IPFS hash values maintained on-chain, which reduces the storage burden greatly.

2.3. Zero-Knowledge Proof (ZKP) System

The ZKP system was first proposed by Goldwasser et al. in 1985. It is an initially interactive protocol between the prover (P) and the verifier (V). With the support of the ZKP system, P could convince V that the statement is true by providing the corresponding proof without revealing the secret (also known as witness) [38,39]. Typically, the ZKP system has properties of completeness, soundness, and zero-knowledgeness [40,41]. The completeness guarantees that any correct statement can always pass the checks of the verifier following

the protocol. The soundness guarantees that any fraud statement cannot cheat the verifier even though P has enough computing power. The zero-knowledgeness guarantees that V cannot learn any more information except for the correctness of the statement [42]. The typical implementation of the ZKP system is the zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK), which is the integration of the multiple cryptographic primitives, such as polynomial commitment and the knowledge of exponent assumption (KEA). Compared with the early implementations of the ZKP system, zk-SNARK has the advantages of succinctness, non-interactiveness, and efficiency, which are realized by relying on the linear probabilistic checkable proof (LPCP) and the common reference string (CRS) model [43,44].

Any NP relation R could be represented as the arithmetic circuit C with the depth $|C|$, which is formed by multiple addition and multiplication gates. The core idea of the zk-SNARK is transforming the circuit satisfaction to the polynomials divisibility by reducing the constraints between gates into a set of polynomials known as the quadratic arithmetic program (QAP). The proof size and the time for proof generation are determined by the circuit depth $|C|$ [42,44]. The construction of the zk-SNARK is depicted in Figure 2, which includes four steps as follows:

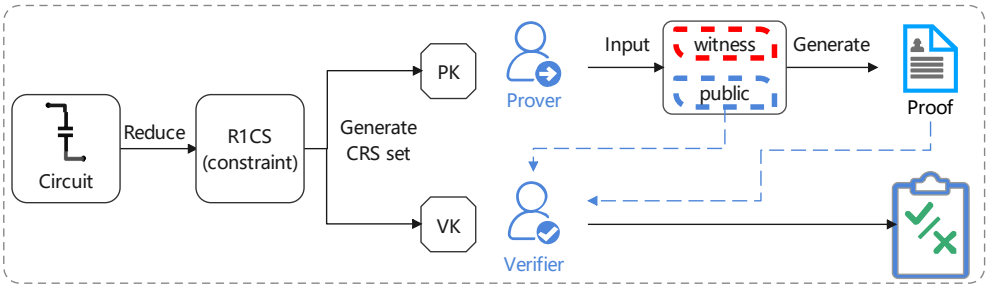

Figure 2. The construction process of the zk-SNARK.

1. Initialization. Converting the NP relation R into the arithmetic circuit C and generating the CRS. Each gate in C is formed by two input wires and one output wire. Different gates could share the same input wires, and the upper gates' output wires can be the lower gates' input wires. The CRS is a set of elements from the finite cyclic groups in the elliptic curves, which are the homophonic hidings of a random secret s. The security of the zk-SNARK is determined by s, which should be discarded completely as "toxic waste" after the CRS generation. The CRS could be accessed by P and V, where the elements for provers and verifiers are known as prover keys (PK) and verifier keys (VK), respectively . It should be noted that the elliptic curves should be pairing-friendly because the proof is validated based on the pairing operations [45].
2. Rank-1 constraints system (R1CS). The R1CS acts as the compiler to help construct the QAP from the arithmetic circuit C. Essentially, the R1CS is a set of n polynomials to represent the relationship among the inputs, outputs, and intermediated values. The construction of the R1CS follows the rule of "one operation and one line". Typically, one operation includes one multiplication gate and serval addition gates (in Groth 16 [46,47]). When the circuit satisfaction holds, the evaluation of the polynomial in each line is zero.
3. Quadratic artchimetic program (QAP). The QAP includes three polynomials, L(x), R(x), and Q(x), to represent the left input wires, right input wires, and output wires of all multiplication gates. Three polynomials satisfy that $Q(x) \bullet \mathbf{v} = (L(x) \bullet \mathbf{v}) \times (R(x) \bullet \mathbf{v})$ and $\mathbf{v} = [1, \mathbf{x}^T, \mathbf{w}^T]^T$, where $\mathbf{x}$ and $\mathbf{w}$ are the public input vector and the witness, respectively. L(x), R(x), and Q(x) represent the constraint of circuit C, which has no relationship with the $\mathbf{v}$ and could be obtained by P and V. Let $\mathbf{Q}(x) = Q(x) \bullet \mathbf{v}$, $\mathbf{L}(x) = L(x) \bullet \mathbf{v}$, and $\mathbf{R}(x) = R(x) \bullet \mathbf{v}$. When all constraints hold, P could generate a polynomial $P(x) = \mathbf{L}(x) \bullet \mathbf{R}(x) - \mathbf{Q}(x)$, and the evaluations for all lines are zeroes, which means that there is a quotient polynomial $\mathbf{H}(x)$ stratifying

that $\mathbf{H}(x) = \mathbf{P}(x)/\mathbf{Z}(x)$. $\mathbf{Z}(x) = \prod_{i=1}^{n}(x-i)$ is known as the target polynomial and could be obtained by both sides.

4. Proof generation. The QAP converts the circuit stratification into the polynomial disviability, which could be verified efficiently based on the Schwartz–Zippel lemma and the polynomial commitment schemes (such as KZG and IPA [48]). The proof is the homomorphic hiding of the H(s), which is also an element of the finite cyclic group and can be obtained by the linear combination of PK. Correspondingly, the proof could be checked through the pairing operations using VK.

It should be noted that more cryptographic primitives, such as KEA, are introduced to force the provers only to use PK and the same **w** to generate the proof as described in [44]. After construction, the main functions of the zk-SNARK could be described as follows:

1. $Setup(C, 1^{\lambda}) \rightarrow pp$, setting the public parameters pp. λ is the security parameter. $pp = (p,\ e,\ G_1,\ G_2,\ G_T,\ g_1,\ g_2,\ g_T)$, where G_1, G_2, and G_T are the finite cyclic groups with the generators of g_1, g_2, and g_T in the elliptic curves EC_1, EC_2, and EC_T satisfying the bilinear mapping $e : G_1(F_p) \times G_2\left(F_p^k\right) \rightarrow G_T\left(F_p^k\right)$. F_p is the finite field of prime p and F_p^k is the extension field of F_p.
2. $GenKey(pp, C) \rightarrow (PK, VK)$, generating the CRS corresponding to the circuit C, where PK and VK are elements for proof generation (in P side) and proof verification (in V side).
3. $GenProof(PK, \mathbf{x}, \mathbf{w}) \rightarrow \pi$, generating the ZKP proof in P side. The algorithm inputs the PK, the public input $mathbfx$, and witness $mathbfw$.
4. $VerProof(VK, \mathbf{x}, \pi) \rightarrow (0, 1)$, verifying the proof. When $C(\mathbf{x},\ \mathbf{w}) = 0$ holds, the algorithm outputs 1, meaning the statement is correct. Otherwise, the algorithm outputs 0, meaning the statement is a fraud.

zk-SNARK is mainly used in the proving statement on private data, anonymous authorization, anonymous payments, outsourcing calculations, etc.

3. Design of the TrsutDFL Framework

This section describes the design of the TrustDFL framework in detail. The overview of the TrustDFL framework is given in Section 3.1, where the components and node roles are introduced. The workflow of the TrustDFL framework is given in Section 3.2. The proof generation for model training and aggregation are introduced in Sections 3.3 and 3.4, respectively. For clarity, symbols used in the paper and their meanings are listed in Table 1.

Table 1. Main parameters and meaning of TrustDFL.

Symbol	Meaning	Symbol	Meaning
TX_U	transaction about local training	TX_A	transaction about model aggregation
PK_T	prover key of local training	PK_A	prover key of model aggregation
VK_T	verifier key of local training	VK_A	verifier key of model aggregation
π_T	ZKP proof of local training	π_A	ZKP proof of model aggregation
M_0	original global model	$\mathrm{H}_0^{\mathrm{M}}$	IPFS hash of the original global model
∇w_i^{jk}	weight gradient of the k-th node in $i-1$-th layer connecting the j-th node in i-th layer	∇b_i^j	bias gradient of the j-th node in i-th layer
t^j	the label	$\boldsymbol{\omega}$	a vector set
$\boldsymbol{\alpha}$	a vector of intermediate variables	$\boldsymbol{\beta}$	a vector of intermediate variables
$\mathbf{o}$	a vector of output variables	$\mathbf{v_1}$	a vector to constrain outputs
$\mathbf{v_2}$	a vector to constrain left inputs	$\mathbf{v_3}$	a vector to constrain right inputs
M_{r-1}	global model of r-th round to be trained	Q_r^k	the k-th updated local model of r-th round model training

3.1. System Scenario

The TrustDFL is a ZKP-based decentralized federated learning framework assisted by the blockchain and smart contracts, where the ZKP provides zero-knowledge proofs for the validity of model training and aggregation with acceptable computational and communication overhead. The blockchain acts as the trust anchor of the multi-party cooperation scenario by recording all data from the participants distributedly in the form of transactions, and the smart contracts are applied for ZKP proof verification. As shown in Figure 3, the TrustDFL framework is an overlay network composed of the blockchain, DFL, and IPFS. The DFL is based on the blockchain, which means the blockchain is formed by the DFL's participants (trainers and aggregators) instead of the third-party infrastructure. The data exchanges in the DFL are in the form of transactions and are involved in the consensus process. Moreover, as an off-chain storage protocol, the IPFS is introduced to alleviate the storage burden brought by the global and local models. All data exchanged in the DFL could be dumped into the IPFS with the IPFS hashes maintained on-chain as the commitments and indices.

In TrustDFL, the trainers and aggregators are almost the same as in the original DFL. The only difference is that the trainers and aggregators in TrustDFL have to attach the trained or aggregated results with the corresponding ZKP proofs. The TrustDFL workers act as the blockchain miners, responsible for the consensus process and the block proposal. In each round of training, each trainer will initiate a transaction including the proof of this round of local training and the hash value of the training results, which is broadcasted and verified by all workers and recorded in the blockchain. After the number of local training transactions on the blockchain reaches a threshold, the trainer collects relevant transactions and extracts IPFS hashes, then retrieves the updated local models from IPFS and aggregates them using the aggregation algorithm. The aggregator initiates a transaction containing the ZKP proof of the aggregation process and the IPFS hash of the updated global model. The global model contained in the first valid aggregated transaction recorded on the blockchain will be used for the next round of training.

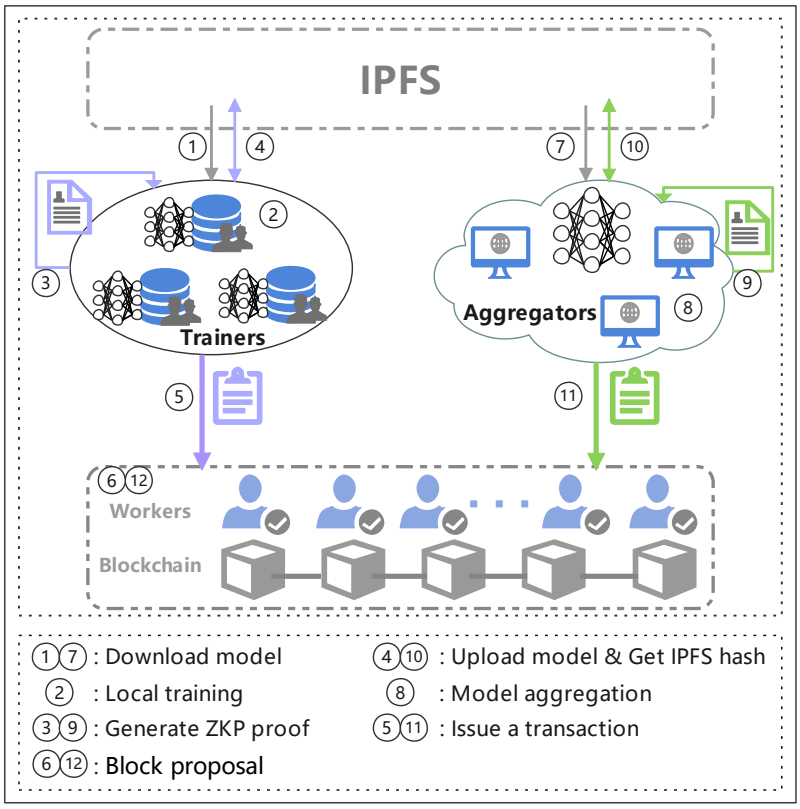

Figure 3. The roles and components of the ZKP-based TrustDFL framework assisted by blockchain.

3.2. System Workflow

Before the training rounds, the system should be initiated by defining the public parameters, including the pairing-friendly elliptic curves (EC_1, EC_2, and EC_T), their generator (g_1, g_2, and g_T), the ZKP implementations (such as Groth 16, Plonk, or STARKs), and the algorithms for the model aggregation (such as FedAvg). For the construction of the ZKP, two CRS sets (CRS_T and CRS_A) should be generated in advance, where $CRS_T = [PK_T, VK_T]$ is for the model training, and $CRS_A = [PK_A, VK_A]$ is for the model aggregation. Typically, the CRS generation and maintenance are delegated to the third party, which could also be realized by federated computation with the support of the smart contracts in TrustDFL. Depending on the selected ZKP implementation, the CRS_T and CRS_A could be the same. For example, since the ZKP implementations, such as Plonk, Sonic, and SuperSonice, could provide the universal and updatable structure reference strings (SRS) composed by the group elements independent of the specific circuits, only one CRS set with enough elements is necessary [49,50]. Moreover, since the neural network consists of multiple layers related to different operations, one CRS should be generated when Groth 16 is used as the ZKP scheme for one operation [46]. However, the CRS generation and maintenance are out of the scope of this paper. For simplicity and without the loss of generality, we assume that all necessary CRS sets are ready before the training rounds begin.

The workflow in TrustDFL could be described as follows:

1. Model publication. In TrustDFL, any node could publish the training task. The original global model M_0 is defined by the architecture of the model to be training (the number of layers, the activation functions, the size of the inputs, etc.), the initialized weights, and some super parameters (epoch, learning rate, etc.). The node first submits all the data to the IPFS and issues a special transaction TX_P with the received IPFS hash $\mathrm{H}_0^{\mathrm{M}}$. The TX_P has the same fields as the normal transactions, but the payload is filled with the IPFS hash. After the TX_P is confirmed in the blockchain, all trainers can learn the training task, extract the IPFS hash, and retrieve the model parameters from the IPFS.
2. Model training. Using the local dataset, all trainers update the model weights based on the retrieved M_0, where the BP process will be repeated according to the defined epoch (step 2). After all iterations are complete, the trainer generates the corresponding ZKP proof π_T based on the PK_T for the validity of the trained results (step 3), where the original weights act as the public inputs, and the local training dataset acts as the witness. In this way, the validity of the local training could be proved without any leakage of the local sensitive information. Then, the trainer submits all data (trained results) into the IPFS and issues a transaction TX_U with the returned IPFS hash and ZKP proof filled in the payload field (steps 4 and 5). Like normal transactions, the TX_U is broadcasted and verified by all workers, which means the contributions of all trainers are recorded in the blockchain.
3. Block proposal. Any workers receiving the TX_U should check its validity. In addition to the signature check, the worker should check the validity of the attached proof π_T by calling the smart contract SC_1 (step 6). The SC_1 implements the verification function $VerProof()$ described in Section 2.3, which should contain the VK_T from the CRS_T and be deployed in advance. Only the TX_U attached with the valid proof could be persisted in the blockchain, which is ready for the model aggregation by the aggregators.
4. Model aggregation. All trained results are accessible for the aggregators when all TX_U are confirmed in the blockchain. The aggregator could collect all the related TX_U by traversing the blockchain, then extract the IPFS hashes and retrieve the updated parameters from the IPFS (step 7). Based on the collected data, the aggregator performs the defined aggregation algorithm and generates the updated global model parameters for the training in the next rounds (step 8). Then, the aggregator generates the corresponding ZKP proof π_A based on the PK_A from the CRS_A, and submits all data into the IPFS (step 9). Finally, the aggregator issues the transaction TX_A with the payload field filled by the returned IPFS hash and ZKP proof (steps 10 and 11).

5. Block proposal. After receiving the TX_As, the worker checks their validity by checking the attached ZKP proof and queues them in an orderly manner. It should be noted that though all valid aggregated results are recorded on-chain, only the results included in the first valid TX_A will be used as the global parameters for the training in the next round.

3.3. ZKP Proof Generation for the Local Training Process

For simplicity and without the loss of generality, we use the MLP as the model to be trained, composed of an input layer, an output layer, and a hidden layer. The activation functions used in hidden and output layers are determined as ReLU and sigmoid, respectively. Without consideration of the activation functions, all operations of each layer are the linear combinations between the inputs and weights, which could be efficiently converted into the arithmetic circuit composed of only addition and multiplication gates. The float point numbers and the nonlinear operations introduced by the activation functions could be expressed as fractions and exponents, which could be mapped to their bit representations and converted to the arithmetic circuits [42]. In this section, we generate the ZKP proof for each layer and finally aggregate them with the support of the Lego SNARK [51]. The ZKP proof aggregation process for each round of local training is shown in Figure 4. Among them, the ZKP proofs at the s-th epoch can be aggregated into c_s, including the proofs of calculation at each layer of forward propagation $(b_{s1}, b_{s2}, \ldots, b_{sp})$ and proofs of calculation between each layer of backpropagation $(b_{s1}, b_{s2}, \ldots, b_{sp})$. ZKP proofs $(c_1, c_2, \ldots, c_s)$ from multiple epochs can also be aggregated into the final ZKP proof π.

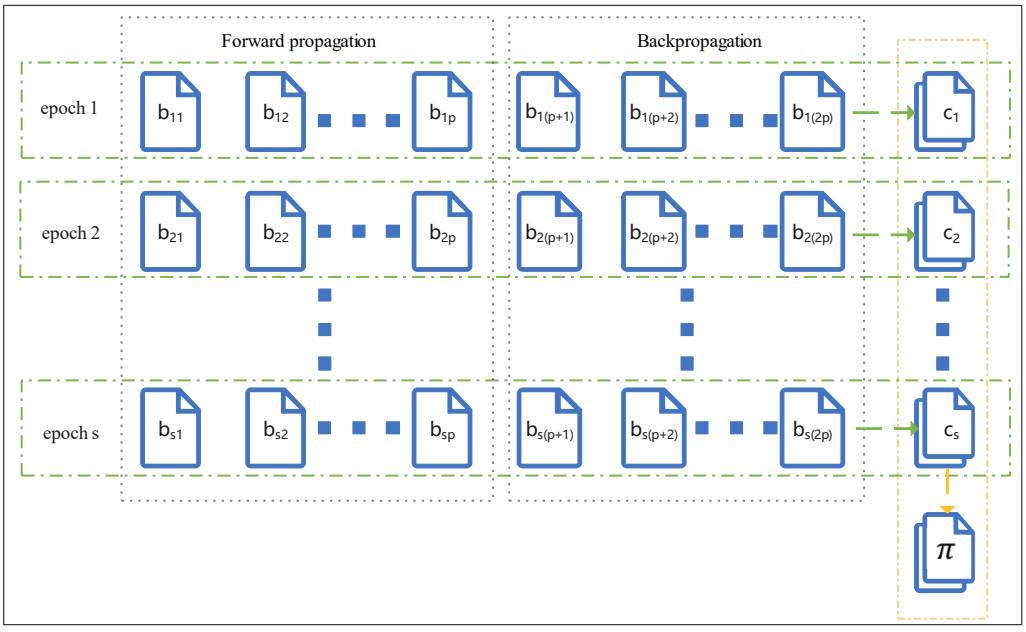

Figure 4. The aggregation process of ZKP proofs for each round of local training.

The FP process can be proved via Groth 16 as described in [19,20,24], which is not the concern of this paper. To prove the validity of the training result, the BP process must be proposed, which is the process of chain rules as shown in Section 2.1. We can get the gradient calculation formula as follows:

$$\begin{cases} \nabla w_i^{jk}(s) = \delta_i^j(s) y_{i-1}^k(s) \\ \nabla b_i^j(s) = \delta_i^j(s) \end{cases} \tag{7}$$

$$\delta_i^j(s) = \begin{cases} f'\left(u_i^j\right) \cdot \sum\limits_{h \in I_{i+1}} \delta_{i+1}^h w_{i+1}^{hj}, \ hidden\ layer \\ f'\left(u_i^j\right) \cdot \left(\frac{1}{n} \sum\limits_{j=1}^{n} \left(y_i^j - t^j\right) y_{i-1}^k\right), \ output\ layer \end{cases} \tag{8}$$

where n and s are the numbers of the samples and epochs, respectively; w_i^{jk} represents the weight of the k-th node in $i-1$-th layer connecting the j-th node in i-th layer; b_i^j refers to the bias of the j-th node in i-th layer; t^j is the label; and $u_i^j = \sum_{k=1}^{p} w_i^{jk} y_{i-1}^k + b_i^j$ (when $i = 2$, y_{i-1}^k is the k-th node in the input layer).

From Equations (7) and (8), we can know that the calculation of gradient involves linear operations and nonlinear operations. Nonlinear operations are mainly the calculation of activation functions and their derivatives. It can be seen from Groth 16 that nonlinear operations need to be converted into linear operations to generate zero-knowledge proofs. We first describe the processing of nonlinear operations and then express the operation process of generating zero-knowledge proofs from linear operations.

3.3.1. Processing Of Nonlinear Operations

The activation functions mainly considered in this article are ReLU and Sigmoid functions, which are expressed as Equation (1) and Equation (2), respectively. For Equation (1), we use the polynomial proposed in the paper by [52] for approximation. In the interval $I = (-a, a)$, our polynomial activation function is given by

$$f_{ReLU}(x) = x^2 + ax \tag{9}$$

For Equation (2), it is often used in the output layer to complete the classification task. We use the following Taylor series to express.

$$f_{sig}(x) = \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3 + \frac{1}{480}x^5 - \frac{17}{80640}x^7 + O\left(x^8\right) \tag{10}$$

The derivative of the Sigmoid function is shown in Equation (11), and the derivative of the ReLU function is shown in Equation (12).

$$f'_{sig}(x) = f_{sig}(x)\left(1 - f_{sig}(x)\right) \tag{11}$$

$$f'_{ReLU}(x) = \begin{cases} 1, \ x > 0 \\ 0, \ x < 0 \end{cases} \tag{12}$$

For Equation (11), there is only one nonlinear function, the Sigmoid function. Therefore, the polynomial approximation of Equation (11) can be realized with the help of Equation (10) and expressed as

$$f'_{sig}(x) = \frac{1}{4} + \frac{1}{4}x - \frac{1}{16}x^2 - \frac{1}{48}x^3 + \frac{1}{480}x^5 + \frac{1}{2304}x^6 - \frac{17}{80640}x^7 + O\left(x^8\right). \tag{13}$$

It can be seen from Equation (12) that the derivative of ReLU is not a smooth function, so we first use Equation (14) to perform a smooth approximation.

$$f'_{ReLU}(x) = \lim_{k \to \infty} \frac{1}{2}(1 + \tanh(kx)) \tag{14}$$

Then, we use Taylor series Equation (15) to perform polynomial approximation of the nonlinear function.

$$f'_{ReLU}(x) = \lim_{k \to \infty} \frac{1}{2}\left(1 + kx - \frac{1}{3}(kx)^3 + \frac{1}{5}(kx)^5 - \frac{1}{7}(kx)^7 + O\left((kx)^8\right)\right) \tag{15}$$

3.3.2. The Operation Process of Linear Calculation to Generate ZKP Proof

When i is the output layer, from Equation (7) and Equation (8) we can see that the gradient calculation formula is $\nabla w_i^{jk} = f'\left(u_i^j\right) \cdot \left(\frac{1}{n}\sum_{j=1}^{n}\left(y_i^j - t^j\right)y_{i-1}^k\right) \cdot y_{i-1}^k$ and establish R1CS for it as follows:

$$\langle \mathbf{v_1}, \boldsymbol{\omega} \rangle = \langle \mathbf{v_2}, \boldsymbol{\omega} \rangle \bullet \langle \mathbf{v_3}, \boldsymbol{\omega} \rangle. \tag{16}$$

Among them, $\langle \cdot, \cdot \rangle$ represents the dot product of vectors. $\boldsymbol{\omega}$ is the vector set of the input variable vector, output variable vector of the equation, and intermediate variable vector ($\boldsymbol{\omega} = [1, \mathbf{t}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\beta}, \mathbf{f}, \mathbf{o}]^T$). $\mathbf{t}$, $\mathbf{y}$, $\boldsymbol{\gamma}$, and $\mathbf{f}$ are all vectors of input variables of the equation ($\mathbf{t} = [t_1, t_2, t_3, ..], \mathbf{y} = [y_1, y_2, y_3, \ldots], \boldsymbol{\gamma} = [\gamma_1, \gamma_2, \gamma_3, \ldots], \mathbf{f} = [f_1, f_2, f_3, \ldots]$). $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are vectors of intermediate variables, and $\mathbf{o}$ is a vector of output variables. $\mathbf{v_1}$, $\mathbf{v_2}$, and $\mathbf{v_3}$ are vectors, usually composed of 0 and 1, used to constrain the outputs, left inputs, and right inputs, respectively.

When i is the output layer, the gradient calculation of the weights involved usually contains multiple calculation equations (multiple inputs and outputs), so they can be generalized by Equation (16) and establish R1CS as follows:

$$V_1 \cdot \boldsymbol{\omega} = (V_2 \cdot \boldsymbol{\omega}) \circ (V_3 \cdot \boldsymbol{\omega}) \tag{17}$$

where $\circ$ denotes the Hadamard product and V_1, V_2, and V_3 are matrices. For R1CS as shown in Equation (17), we can use Lagrangian interpolation method to convert it into QAP as follows:

$$(A(x) \cdot \boldsymbol{\omega}) \circ (B(x) \cdot \boldsymbol{\omega}) - C(x) \cdot \boldsymbol{\omega} = 0 \tag{18}$$

where $A(x)$, $B(x)$, and $C(x)$ are matrices expressed as $A(x) = [A_1(x), A_2(x), A_3(x), \ldots]$, $B(x) = [B_1(x), B_2(x), B_3(x), \ldots]$, and $C(x) = [C_1(x),\ C_2(x),\ C_3(x), \ldots]$, respectively.

In the same way, when i is the hidden layer, R1CS and QAP can be constructed. It should be noted that the proofs for each layer are independent from each other, which could furthermore be merged to a single one by Lego SNARK. Recursively, the proofs for all epochs can also be merged by Lego SNARK. Finally, the training result could be prosed by the merged single proof, which could be checked by validators calling the function $VerProof()$.

3.4. ZKP Proof Generation for the Model Aggregation Process

During the r-th round of model training, the global model to be trained is M_{r-1} and multiple trainers engage in local training. The Federated Averaging (FedAvg) algorithm is used to aggregate the local models and obtain an updated global model denoted as M_r. The aggregators obtain K updated local model IPFS hashes from the blockchain and then access IPFS to obtain K updated local models. The k-th updated local model is Q_r^k, and the updated global model representation is shown in Equation (19).

$$M_r = \sum_{k=1}^{K} \frac{n_k}{n} Q_r^k \tag{19}$$

Based on Equation (19), it is clear that the computations involved in the aggregation algorithm are linear. Based on Equation (16), the R1CS could be constructed to obtain the QAP by homomorphism between the vector and polynomial rings.

4. System Analysis

This section provides an analysis of the TrustDFL system. The security objectives of the TrustDFL framework are given in Section 4.1. We prove that TrustDFL achieves all design goals, namely trustworthiness, resistance to attacks, data privacy, scalability, and decentralization in Sections 4.2–4.6, respectively.

4.1. Security Objectives

The specific goals of proposed system design are as follows.

- Trustworthiness: The trustworthiness guarantees the correctness of local model and updated global model, which mainly depends on the correct execution of local training and aggregation process. If the node does not perform the above process as intended, it will fail validation and be discarded. In addition, any node cannot tamper with the model written to the blockchain through effective means.
- Resistance to attacks: TrustDFL can resist to model poisoning attacks and collusion attacks, i.e., compromised nodes cannot upload invalid updates (local model or global model) to hinder the convergence of the model and cannot cooperate to manipulate verification results to attack the model.
- Data privacy: The trainer can prove to others that he has conducted the local training process honestly without leaking local data.
- Scalability: TrustDFL uses IPFS to reduce the pressure of on-chain storage and uses zk-SNARK with very small proof size and on-chain verification complexity.
- Decentralization: TrustDFL can avoid single point of failure and the communication traffic congestion caused by the central server. The overall DFL task is accomplished by nodes without any centralized server and the trusted third party.

4.2. Trustworthiness

TrustDFL uses zk-SNARK to ensure the trustworthiness of the local model and the updated global model. zk-SNARK has perfect completeness and can verify the integrity of the calculation process. This ensures reliable execution of the local training process and model aggregation process. Honest provers extract relevant matrix information that is accurate (with valid witness) and generate proofs π using the proof circuit $C(\mathbf{x}, \mathbf{w})$ and the prover key PK. When verifying the local training process, the relevant matrix information includes the initial weight values, inputs, and updated local models, etc. When verifying the model aggregation process, the relevant matrix information encompasses K local models and the updated global model, etc. Honest verifiers utilize the verifier key VK, proof π, and public input $\mathbf{x}$ to obtain a verification result of 1. For any $(\mathbf{x}, \mathbf{w})$ satisfying $C(\mathbf{x}, \mathbf{w}) = 0^l$, we have:

$$Pr\left(Verproof(VK, \mathbf{x}, \pi) = 1 \middle| \begin{matrix} Setup(1^\lambda, C) \rightarrow (EK, VK) \\ Genproof(PK, \mathbf{x}, \mathbf{w}) \rightarrow \pi \end{matrix}\right) = 1. \tag{20}$$

In addition, our solution chooses to the transaction with the payload field filled by the returned IPFS hash and ZKP proof to establish trust relationships between distributed nodes. Once transaction are verified and written into the blockchain by reaching a consensus, no one can modify or deny the data. Moreover, it is impossible to forge transactions verified by miners, and it is impossible to forge the change record of the entire transaction.

4.3. Resistance to Attacks

The TrustDFL can effectively resist model poisoning attacks. The soundness of zk-SNARK means that the probability of success for any PPT (Probabilistic Polynomial Time) malicious attacker algorithm is negligible. For every probabilistic polynomial-time adversary $\mathcal{A}$, there exists a probabilistic polynomial-time witness extractor $\mathcal{E}$ such that:

$$Pr\left(\begin{matrix} C(\mathbf{x}, \mathbf{w}) \neq 0^l \\ Verproof(VK, \mathbf{x}, \pi) = 1 \end{matrix} \middle| \begin{matrix} Setup(1^\lambda, C) \rightarrow (PK, VK) \\ \mathcal{A}(PK, VK) \rightarrow (\mathbf{x}, \pi) \\ \mathcal{E}(PK, VK) \rightarrow \mathbf{w} \end{matrix}\right) \leq negl(\lambda). \tag{21}$$

The soundness of zk-SNARK ensures that the local model and the updated global model are not forged. If the verification equation of the local proof (aggregation proof) holds, it indicates that the node correctly performed the local training task (the task of model aggregation). Therefore, it can effectively resist model attacks.

Furthermore, our scheme can effectively resist collusion attacks. zk-SNARK is non-interactive, so the proof can be attached to the blockchain transaction, and the smart contract implements on-chain verification. This effectively prevents malicious provers and malicious verifiers from colluding to forge wrong proof and verification results to attack the model. Hence, only when the number of malicious nodes exceeds 50% of the total system nodes, the probability of a successful attack becomes great. Malicious nodes would require 51% of the computational resources to attack the system, which incurs costs far exceeding the benefits.

4.4. Data Privacy

The zero-knowledge property of zk-SNARK means that for any probabilistic polynomial-time adversaries $\mathcal{A}$ and every input $\mathbf{x}$ of a given circuit C, there exists a simulator S such that the following two probabilities are approximately equal:

$$Pr\left(\mathcal{A}(PK, VK, \pi) = 1 \middle| \begin{matrix} Setup(1^{\lambda}, C) \rightarrow (PK, VK) \\ Genproof(EK, \mathbf{x}, \mathbf{w}) \rightarrow \pi \end{matrix}\right), \tag{22}$$

$$Pr\left(\mathcal{A}(\hat{PK}, \hat{VK}, \pi) = 1 \middle| \begin{matrix} S(C) \rightarrow (\hat{PK}, \hat{VK}, trap) \\ S(\hat{PK}, \mathbf{x}, trap) \rightarrow \pi \end{matrix}\right). \tag{23}$$

Therefore, the verifier can complete the verification of the training process without knowing the original data, ensuring the security of local data.

4.5. Scalability

TrustDFL introduces zero-knowledge proof and IPFS. While ensuring the safety and reliability of the model, the model is stored on IPFS and the hash is written into the blockchain, which reduces the storage pressure on the chain. Although the Groth 16 solution needs to generate different CRS for different operations, compared with Plonk, Sonic, etc., it maintains optimal performance in terms of verification workload and proof size. Its verification calculation complexity is low and has nothing to do with the circuit complexity. It only needs to verify a paring product equation. There are only three pairing calculations in this equation. The proof size is small and basically constant, and only contains three group elements. Our solution, based on Groth 16's zk-SNARK, performs a computationally intensive and complex circuit and proof generation process off-chain, while performing a simple and fast proof verification process on-chain. Compared with the solution that uses the blockchain consensus mechanism to implement model verification, e.g., [17,18], our solution has higher scalability.

4.6. Decentralization

The decentralization is achieved without any central server. In TrustDFL, the aggregation work calculated by the central server is distributed to multiple nodes. All nodes are equal and contribute to the overall functionality, thereby mitigating the risks associated with single points of failure.

5. Implementation and Evaluation

This section realizes the Proof-of-Concept implementation of the proposed TrustDFL architecture, where its performance is estimated in terms of global model accuracy and overheads introduced by the ZKP schemes in Section 5.1 and Section 5.2, respectively. All estimations run in a Linux laptop with Ubuntu 18.04 LTS operation system, RAM of 16 GB, and CPU of Intel(R) Core(TM) i7-8665@2.11 GHz. The implementation overview of the TrustDFL is as follows:

1. The decentralized federated learning architecture is deployed based on PyTorch (Version 1.6.0), where the global model to be trained is determined as MLP, and the classic datasets of MNIST, Fashion-MNIST, CIFAR10, and CIFAR100 are determined as the training sets. The stochastic gradient descent algorithm is used for local model training.

2. The specific private blockchain is implemented following the basic design of Ethereum. Only the core functions, such as consensus algorithm data persistence, are realized for simplicity. The consensus algorithm is determined as PoW, and the difficulty is set to 0×1 for efficiency. The smart contracts are simulated by C++ scripts. All participants of DFL and blockchain are simulated by the multiple processes, and the IPFS is accessed via the web entrance provided by the Protocol Lab.
3. Specifically, the samples in MNIST and Fashioin-MNIST are grayscale images of 28×28, where 60,000 and 10,000 samples are for training and testing, respectively. The samples in CIFAR10 and CIFAR100 are 32×32 colorful images, where 50,000 and 10,000 samples are for training and testing, respectively.
4. The ZKP scheme used in the TrustDFL is determined as Groth 16, which provides a smaller witness size and faster proof generation/validation. The Groth 16 is implemented based on the C++ library (libSNARK).
5. Limited to the experiment environment, the DFL and blockchain system are formed by ten nodes, where two nodes are chosen randomly as the malicious nodes to launch the model poisoning attacks by publishing the randomly generated float-point numbers as the local training results.

5.1. Accuracy of the Global Model to Be Trained

The target of the DFL is obtaining a converged global model to provide prediction service for external users. Thus, the accuracy of the global model is the major indicator to measure the model's effectiveness. In this section, the estimation process includes two phases based on the ten participants. In phase one, eight honest nodes execute the stochastic gradient descent based on the local datasets and publish the trained result, where the two malicious nodes only publish randomly generated float-point numbers. The designated aggregators aggregate the received trained results based on the FedAvg algorithm and update the global model for the next round of training. In phase two, all participants should publish the local training results tighter with the corresponding ZKP proofs, and all participants act as the aggregators by validating the correctness of the ZKP proofs via the smart contracts. The two phases will run based on the datasets of MINST, Fashion-MNIST, CIFAR10, and CIFAR100 in subsequence. For each phase, 10% of samples in the training sets are dispatched randomly to each node. For simplicity, the training set on each node is mutually exclusive. Two phases share the same learning ratios, batch size, and epochs.

For each phase, the training process is repeated ten times. Then, based on the testing sets, all prediction results are records, where the proportion of the results matching the preset labels to the data size is defined as the accuracy of the global model. Based on the different datasets, the accuracy of the global models obtained in two phases is depicted in Figure 5. As shown in Figure 5, the accuracy of the global model from phase two (with the TrustDFL applied) is higher than the phase one (with FedAvg algorithm for model aggregation). Therefore, the proposed TrustDFL can effectively remove the risk of poisoning attacks caused by malicious nodes and guarantee the accuracy and usability of the model. Moreover, the accuracies for datasets of CIFAR10 and CIFAR100 are lower than the accuracies for the datasets of MNIST and Fashion-MNIST. That is because, in this estimation, the MLP is determined as the global model to be trained, which is unsuitable for handling images with multiple channels. For different datasets, it could be seen that the accuracy of the global model from phase one shows volatility. That is because the aggregator cannot distinguish between the random numbers and honest training results effectively. The randomness from the malicious nodes causes accuracy volatility.

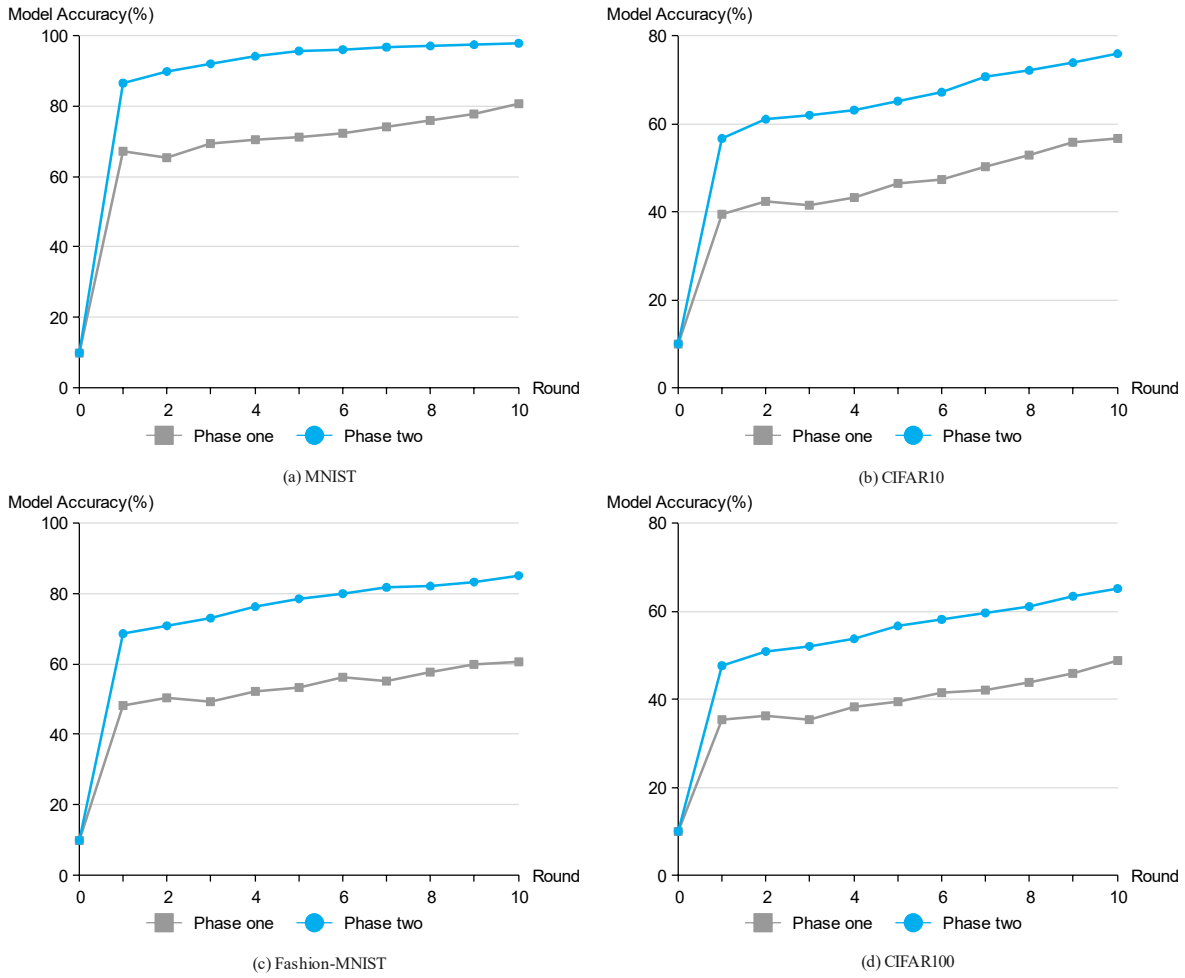

Figure 5. Comparison of the accuracy of models of different schemes on different datasets. (**a**) Comparison of model accuracy of different schemes on the MNIST dataset. (**b**) Comparison of model accuracy of different schemes on the CIFAR10 dataset. (**c**) Comparison of model accuracy of different schemes on Fashion-MNIST dataset. (**d**) Comparison of model accuracy of different schemes on the CIFAR100 dataset.

5.2. Overhead

In this section, we evaluate the storage and time consumption caused by building a zero-knowledge proof; the storage consumption is mainly the proof key size, the verification key size and the proof size; the time consumption is mainly setup time, proving time and verifying time. We build ZKP proofs for different epochs of local training based on the MLP model and adopt the Groth16 scheme in the libSNARK. At the same time, we discussed the experimental results.

As can be seen from Figure 6, the largest storage occupied by ZKP for different numbers of epoch in local training is the prover key, and it is constant, about 150.6 MB. The second one that occupies a larger amount of memory is the verifier key. As shown in Figure 6, it is also constant in different numbers of epoch, about 18.7 MB. The proof size is the smallest and much smaller than prover key and verifier key. The average proof size per epoch is 1.2 KB. For different numbers of epoch, the prover key size and verifier key size are constant. This is because the task publisher will stipulate the architecture of the model to be trained (number of layers, activation function, input size, etc.), and the specified calculation operations performed in each epoch will not change, so the circuit (multiplication gate, adding gates and wires) of each epoch is based on calculations generated will not change. According to Lego SNARK, when generating ZKP proofs for multiple epochs of training, ZKP proofs for each training epoch are constructed and stacked. Therefore, local training

for different epochs only requires the same prover key and verifier key. The number of ZKP proofs will increase linearly with the linear increase in the number of epochs, so the size of the proof will also increase linearly.

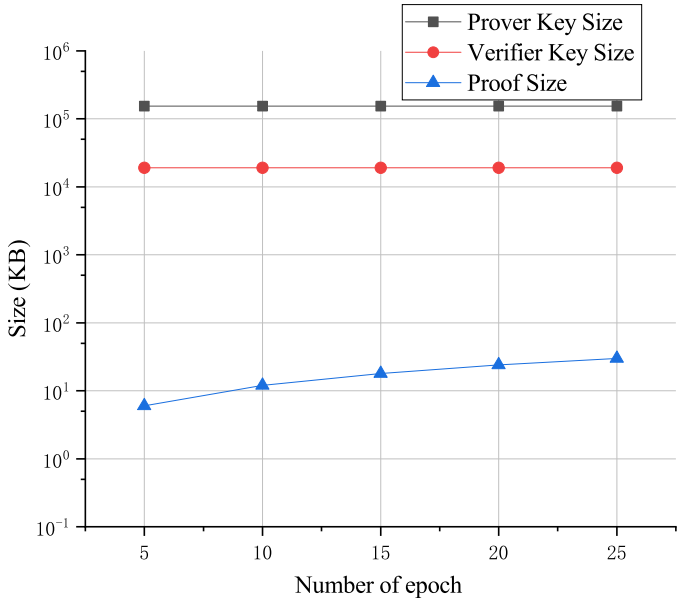

Figure 6. Storage overhead of ZKP for local training with different epoch numbers.

As can be seen from Figure 7, for the time taken to build ZKP for local training with different numbers of epochs, the setup time is the longest and is constant, about 236.7 s. Since the model structure trained in each epoch will not change, and the circuit structure built will not change, the setup time will not change either. In addition, as long as the structure of the trained model is constant, the setup time only needs to be once. The time to build and verify the proof grows with the number of epochs of local training. This is because as the number of epochs increases, the calculations required to prove increase, the number of proofs constructed increases, and the proofs that need to be verified also increase. It can also be seen from Figure 7 that it takes a long time to build the proof, but each node is synchronized off-chain. The time to verify the proof is very short, with an average of 0.17 s per epoch. Therefore, it is appropriate to choose on-chain verification proof, and the performance consumption brought to the blockchain is also considerable.

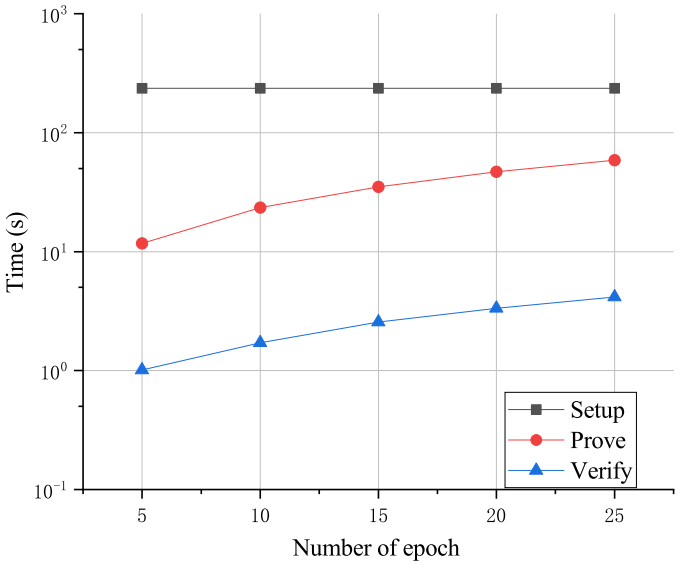

Figure 7. The time consumption of ZKP for local training with different epoch numbers.

5.3. Scheme Comparison

We conducted a functional comparative analysis between the proposed TrustDFL and BFLC, VFChain, PTDFL, and PZKP-FL schemes, as shown in Table 2. BFLC and VFChain use a combination of blockchain consensus mechanism and cross-validation to verify local training results, but this method cannot be applied to all data sets and cannot be used to verify the calculation correctness of model aggregation. When the model and data set are large, the validation time will be long. The verification time of ZKP is independent of the data set and model size. Both PTDFL and PZKP-FL use ZKP to verify the local training results, but PTDFL does not take into account the verification of model aggregation. PTDFL did not choose to store the ZKP proof in the blockchain and could not prevent malicious nodes from colluding to tamper with the verification results. Although PZKP-FL takes the above issues into consideration, its framework is CFL and does not solve the single point of failure problem. TrustDFL not only takes into account the above issues, but also considers the huge storage overhead problem caused by storing the model on the blockchain. We choose to store the IPFS hash on the blockchain and store the model in IPFS, which effectively alleviates the storage pressure on the blockchain.

Table 2. The functionality comparison with the existing schemes.

Schemes	BFLC [17]	VFChain [18]	PTDFL [22]	PZKP-FL [23]	TrustDFL
FL type	DFL	DFL	DFL	CFL	DFL
Verification method	k-fold cross validation	cross validation	ZKP	ZKP	ZKP
Local training verification	✓	✓	✓	✓	✓
Model aggregation verification	-	-	-	✓	✓
Blockchain	✓	✓	-	✓	✓
Storage scalability	-	-	-	-	✓

6. Discussion

Since this study has certain limitation, we will focus on the following three aspects in the future: (1) long proof generation time. This can be solved by improving the proof generation algorithm, specifically by considering converting QAP-based proof generation into QVP (Quadratic vector program) or QMP (Quadratic matrix program) [21]. In addition, from a hardware perspective, the proof generation time can be reduced by using hardware acceleration MSM (Matrix scalar Multiplucation)/NTT (Number Theory Transformation) components [53]. (2) Circuit-specific trusted setup. This can be solved by considering universal setup process, such as the SRS in Plonk and Sonic, or by splitting the entire computation into small and reusable steps/templates as Circom does [54]. (3) Extra communication overhead. This can be improved by more efficient proof aggregation algorithm, such as recursive ZKP, and develop more efficient elliptic curves such as bls12_377 [55].

7. Conclusions

In this paper, TrustDFL is proposed to solve the single point of failure and model attack problems in FL. TrustDFL is a trustworthy and verifiable decentralized federated learning framework based on blockchain and ZKP. It implements computational verification of the local training process and model aggregation process to resist model poisoning attacks and ensure the effectiveness of the global model. The ZKP scheme is used to prove the effectiveness of the local training without leaking sensitive information, and the blockchain acts as the trust anchor in the multi-party cooperation scenarios to record all activities for punishment and rewards. With the support of smart contracts, the verification for local training and model aggregation could be executed automatically. Moreover, the IPFS is introduced to alleviate the storage burden introduced by the models for the blockchain, where only the IPFS hashes of the models are persisted on the blockchain. According to the theoretical analysis and PoC implementation, the proposed TrustDFL framework provides enhanced

security and privacy protection with limited storage overhead. Compared with the conventional FL framework using the FedAvg algorithm, the proposed TrustDFL framework distributes the workloads among all participants, providing higher efficiency for model aggregation and avoiding the single point of failure. Moreover, the TrustDFL can combine with more privacy-protected schemes, such as differential privacy secret sharing schemes, and could be applied for more complicated models, such as CNN and transformers.

Author Contributions: Conceptualization, J.Y. and W.Z.; methodology, W.Z. and Z.G. (Zhaohui Guo); software, J.Y. and W.Z.; validation, J.Y., W.Z. and Z.G. (Zhaohui Guo); formal analysis, J.Y., W.Z. and Z.G. (Zhen Gao); investigation, J.Y., W.Z. and Z.G. (Zhen Gao); writing—original draft preparation, W.Z.; writing—review and editing, Z.G. (Zhaohui Guo), W.Z. and Z.G. (Zhen Gao); project administration, J.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the College Government Procurement Branch of Education Accounting Society of China, grant number EASCCGPB2022MS24.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to the funding restrictions.

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

Abbreviations

The following abbreviations are used in this manuscript:

ZKP	Zero-knowledge Proof
IPFS	Interplanetary file system
Dapp	Develop decentralization applications
zk-SNARK	Zero-knowledge succinct non-interactive argument of knowledge
KEA	Knowledge of exponent assumption
R1CS	Rank-1 constraints system
QAP	Quadratic artchimetic program
CRS	Common reference string
LPCP	Linear probabilistic checkable proof
MLP	Multi-layer perceptron
CNN	Convolutional neural networks
FedAvg	FederatedAveraging
ML	Machine learning
FL	Federated learning
DNN	Deep neural network
MLaaS	Machine learning as a service
CFL	Centralized FL
DFL	Decentralized FL

References

1. Jordan, M.I.; Mitchell, T.M. Machine learning: Trends, perspectives, and prospects. *Science* **2015**, *349*, 255–260. [CrossRef] [PubMed]
2. Javed, A.R.; Ahmed, W.; Pandya, S.; Maddikunta, P.K.R.; Alazab, M.; Gadekallu, T.R. A survey of explainable artificial intelligence for smart cities. *Electronics* **2023**, *12*, 1020. [CrossRef]
3. Shinde, P.P.; Shah, S. A review of machine learning and deep learning applications. In Proceedings of the 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 16–18 August 2018; pp. 1–6.
4. Devi, I.; Karpagam, G.; Kumar, B.V. A survey of machine learning techniques. *Int. J. Comput. Syst. Eng.* **2017**, *3*, 203–212. [CrossRef]
5. Ribeiro, M.; Grolinger, K.; Capretz, M.A. Mlaas: Machine learning as a service. In Proceedings of the 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 9–11 December 2015; pp. 896–990.
6. AbdulRahman, S.; Tout, H.; Ould-Slimane, H.; Mourad, A.; Talhi, C.; Guizani, M. A survey on federated learning: The journey from centralized to distributed on-site learning and beyond. *IEEE Internet Things J.* **2020**, *8*, 5476–5497. [CrossRef]
7. Zhang, C.; Xie, Y.; Bai, H.; Yu, B.; Li, W.; Gao, Y. A survey on federated learning. *Knowl. Based Syst.* **2021**, *216*, 106775. [CrossRef]
8. Yang, Q.; Liu, Y.; Chen, T.; Tong, Y. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol. TIST* **2019**, *10*, 1–19. [CrossRef]

9. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the Artificial Intelligence and Statistics, PMLR, Ft. Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
10. Mothukuri, V.; Parizi, R.M.; Pouriyeh, S.; Huang, Y.; Dehghantanha, A.; Srivastava, G. A survey on security and privacy of federated learning. *Future Gener. Comput. Syst.* **2021**, *115*, 619–640. [CrossRef]
11. Fu, X.; Peng, R.; Yuan, W.; Ding, T.; Zhang, Z.; Yu, P.; Kadoch, M. Federated learning-based resource management with blockchain trust assurance in smart IoT. *Electronics* **2023**, *12*, 1034. [CrossRef]
12. Nguyen, D.C.; Ding, M.; Pathirana, P.N.; Seneviratne, A.; Li, J.; Poor, H.V. Federated learning for internet of things: A comprehensive survey. *IEEE Commun. Surv. Tutorials* **2021**, *23*, 1622–1658. [CrossRef]
13. Blanchard, P.; El Mhamdi, E.M.; Guerraoui, R.; Stainer, J. Machine learning with adversaries: Byzantine tolerant gradient descent. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 104.
14. Yin, D.; Chen, Y.; Kannan, R.; Bartlett, P. Byzantine-robust distributed learning: Towards optimal statistical rates. In Proceedings of the International Conference on Machine Learning. PMLR, Stockholm Sweden, 10–15 July 2018; pp. 5650–5659.
15. Witt, L.; Heyer, M.; Toyoda, K.; Samek, W.; Li, D. Decentral and incentivized federated learning frameworks: A systematic literature review. *IEEE IoT J.* **2022**, *10*, 3642–3663. [CrossRef]
16. Wani, S.; Imthiyas, M.; Almohamedh, H.; Alhamed, K.M.; Almotairi, S.; Gulzar, Y. Distributed denial of service (DDoS) mitigation using blockchain—A comprehensive insight. *Symmetry* **2021**, *13*, 227. [CrossRef]
17. Li, Y.; Chen, C.; Liu, N.; Huang, H.; Zheng, Z.; Yan, Q. A blockchain-based decentralized federated learning framework with committee consensus. *IEEE Netw.* **2020**, *35*, 234–241. [CrossRef]
18. Peng, Z.; Xu, J.; Chu, X.; Gao, S.; Yao, Y.; Gu, R.; Tang, Y. Vfchain: Enabling verifiable and auditable federated learning via blockchain systems. *IEEE Trans. Netw. Sci. Eng.* **2021**, *9*, 173–186. [CrossRef]
19. Fan, Y.; Xu, B.; Zhang, L.; Song, J.; Zomaya, A.; Li, K.C. Validating the integrity of convolutional neural network predictions based on zero-knowledge proof. *Inf. Sci.* **2023**, *625*, 125–140. [CrossRef]
20. Zhao, L.; Wang, Q.; Wang, C.; Li, Q.; Shen, C.; Feng, B. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 2524–2540. [CrossRef]
21. Lee, S.; Ko, H.; Kim, J.; Oh, H. vcnn: Verifiable convolutional neural network based on zk-snarks. *Cryptol. ePrint* **2020**, 584, 1–16.
22. Wang, L.; Zhao, X.; Lu, Z.; Wang, L.; Zhang, S. Enhancing privacy preservation and trustworthiness for decentralized federated learning. *Inf. Sci.* **2023**, *628*, 449–468. [CrossRef]
23. Xing, Z.; Zhang, Z.; Li, M.; Liu, J.; Zhu, L.; Russello, G.; Asghar, M.R. Zero-Knowledge Proof-based Practical Federated Learning on Blockchain. *arXiv* **2023**, arXiv:2304.05590.
24. Heiss, J.; Grünewald, E.; Tai, S.; Haimerl, N.; Schulte, S. Advancing blockchain-based federated learning through verifiable off-chain computations. In Proceedings of the 2022 IEEE International Conference on Blockchain (Blockchain), Espoo, Finland, 22–25 August 2022; pp. 194–201.
25. Reegu, F.A.; Abas, H.; Gulzar, Y.; Xin, Q.; Alwan, A.A.; Jabbari, A.; Sonkamble, R.G.; Dziyauddin, R.A. Blockchain-Based Framework for Interoperable Electronic Health Records for an Improved Healthcare System. *Sustainability* **2023**, *15*, 6337. [CrossRef]
26. Bounds.; Lloyd.; Mathew.; Waddell. A multilayer perceptron network for the diagnosis of low back pain. In Proceedings of the IEEE 1988 International Conference on Neural Networks, San Diego, CA, USA, 24–27 July 1988; pp. 481–489.
27. Taud, H.; Mas, J. Multilayer perceptron (MLP). In *Geomatic Approaches for Modeling Land Change Scenarios*; Springer: Berlin, Germany, 2018; pp. 451–455.
28. Savalia, S.; Emamian, V. Cardiac arrhythmia classification by multi-layer perceptron and convolution neural networks. *Bioengineering* **2018**, *5*, 35. [CrossRef] [PubMed]
29. Underwood, S. Blockchain beyond bitcoin. *Commun. ACM* **2016**, *59*, 15–17. [CrossRef]
30. Zheng, Z.; Xie, S.; Dai, H.; Chen, X.; Wang, H. An overview of blockchain technology: Architecture, consensus, and future trends. In Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress), Boston, MA, USA, 11–14 December 2017; pp. 557–564.
31. Yli-Huumo, J.; Ko, D.; Choi, S.; Park, S.; Smolander, K. Where is current research on blockchain technology?—A systematic review. *PLoS ONE* **2016**, *11*, e0163477. [CrossRef] [PubMed]
32. Zou, W.; Lo, D.; Kochhar, P.S.; Le, X.B.D.; Xia, X.; Feng, Y.; Chen, Z.; Xu, B. Smart contract development: Challenges and opportunities. *IEEE Trans. Softw. Eng.* **2019**, *47*, 2084–2106. [CrossRef]
33. Peters, G.W.; Panayi, E. *Understanding Modern Banking Ledgers through Blockchain Technologies: Future of Transaction Processing and Smart Contracts on the Internet of Money*; Springer: Berlin, Germany, 2016.
34. Dannen, C. *Introducing Ethereum and Solidity*; Springer: Berlin, Germany, 2017; Volume 1.
35. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15.
36. Benet, J. Ipfs-content addressed, versioned, p2p file system. *arXiv* **2014**, arXiv:1407.3561.
37. Zheng, Q.; Li, Y.; Chen, P.; Dong, X. An innovative IPFS-based storage model for blockchain. In Proceedings of the 2018 IEEE/WIC/ACM international conference on web intelligence (WI), Santiago, Chile, 3–6 December 2018; pp. 704–708.

38. Fiege, U.; Fiat, A.; Shamir, A. Zero knowledge proofs of identity. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 1987 ; pp. 210–217.
39. Kilian, J. A note on efficient zero-knowledge proofs and arguments. In Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, Victoria, BC, Canada, 4–6 May 1992; pp. 723–732.
40. Cramer, R.; Damgård, I.; MacKenzie, P. Efficient zero-knowledge proofs of knowledge without intractability assumptions. In Proceedings of the International Workshop on Public Key Cryptography, Melbourne, Australia, 18–20 January 2000; pp. 354–372.
41. Ben-Sasson, E.; Chiesa, A.; Tromer, E.; Virza, M. Succinct {Non-Interactive} zero knowledge for a von neumann architecture. In Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14), San Diego, CA, USA 20–22 August 2014; pp. 781–796.
42. Parno, B.; Howell, J.; Gentry, C.; Raykova, M. Pinocchio: Nearly practical verifiable computation. *Commun. ACM* **2016**, *59*, 103–112. [CrossRef]
43. Abe, M.; Fehr, S. Perfect NIZK with adaptive soundness. In Proceedings of the Theory of Cryptography Conference, Amsterdam, The Netherlands, 21–24 February 2007; pp. 118–136.
44. Gennaro, R.; Gentry, C.; Parno, B.; Raykova, M. Quadratic span programs and succinct NIZKs without PCPs. In Proceedings of the Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, 26–30 May 2013; pp. 626–645.
45. Bowe, S.; Gabizon, A.; Miers, I. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. *Cryptol. ePrint* **2017**, *2017*, 1–24.
46. Groth, J.; Maller, M. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 20–24 August 2017; pp. 581–612.
47. Groth, J. On the size of pairing-based non-interactive arguments. In Proceedings of the Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, 8–12 May 2016; pp. 305–326.
48. Buterin, V. Quadratic Arithmetic Programs: From Zero to Hero. 2016. Available online: https://medium.com/VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649 (accessed on 17 March 2023).
49. Maller, M.; Bowe, S.; Kohlweiss, M.; Meiklejohn, S. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 2111–2128.
50. Gabizon, A.; Williamson, Z.J.; Ciobotaru, O. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptol. ePrint* **2019**, *953*, 1–34.
51. Campanelli, M.; Fiore, D.; Querol, A. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 2075–2092.
52. Ali, R.E.; So, J.; Avestimehr, A.S. On polynomial approximations for privacy-preserving and verifiable relu networks. *arXiv* **2020**, arXiv:2011.05530.
53. Zhang, Y.; Wang, S.; Zhang, X.; Dong, J.; Mao, X.; Long, F.; Wang, C.; Zhou, D.; Gao, M.; Sun, G. Pipezk: Accelerating zero-knowledge proof with a pipelined architecture. In Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 14–18 June 2021; pp. 416–428.
54. Bellés-Muñoz, M.; Baylina, J.; Daza, V.; Muñoz-Tapia, J.L. New privacy practices for blockchain software. *IEEE Softw.* **2021**, *39*, 43–49. [CrossRef]
55. El Housni, Y.; Guillevic, A. Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. In Proceedings of the International Conference on Cryptology and Network Security, Virtual, 14–16 December 2020; pp. 259–279.