Blockchain as a Service: A Decentralized and Secure Computing Paradigm

Gihan J. Mendis*, Yifu Wu*, Jin Wei*, Moein Sabounchi*, and Rigoberto Roche'†

*Department of Electrical and Computer Engineering
University of Akron, Akron, Ohio
†NASA Glenn Research Center, Cleveland, Ohio

Abstract—**Thanks to the advances in machine learning, data-driven analysis tools have become valuable solutions for various applications. However there still remain essential challenges to develop effective data-driven methods because of the need to acquire a large amount of data and to have sufficient computing power to handle the data. In many instances these challenges are addressed by relying on a dominant cloud computing vendor, but, although commercial cloud vendors provide valuable platforms for data analytics, they can suffer from a lack of transparency, security, and privacy-perservation. Furthermore, reliance on cloud servers prevents applying big data analytics in environments where the computing power is scattered. To address these challenges, a decentralize, secure, and privacy-preserving computing paradigm is proposed to enable an asynchronized cooperative computing process amongst scattered and untrustworthy computing nodes that may have limited computing power and computing intelligence. This paradigm is designed by exploring blockchain, decentralized learning, homomorphic encryption, and software defined networking(SDN) techniques. The performance of the proposed paradigm is evaluated via different scenarios in the simulation section.**

Index Terms—**Blockchain, Decentralized and Secure Learning, Machine Learning, Privacy, Security**

I. INTRODUCTION

Due to the advances in sensing and computing, data-driven methods, such as machine learning techniques [1], have become very promising solutions for different applications [2]–[4]. However, there are two essential challenges in implementing these techniques: (1) acquisition of large amount of data, and (2) requirement of enough computing power, which enforces the reliance on a dominant cloud computing vendor. Although the cloud servers provide valuable platforms for big data analytics, there remain several essential challenges in adopting the commercial cloud vendors: (1) transparency, (2) security, and (3) privacy [5]. With the rise of awareness for data privacy, end-users have become reluctant to share their data. Also, in another perspective, user data are becoming a valuable asset. In some domains, such as the healthcare sector, federal and civil service offices, there is an abundance of valuable data, however, due to privacy laws and regulations, these data cannot be shared with the third party. Furthermore, the reliance on cloud servers also limit the potentials of applying big data analytics for the environment where the computing power is scattered. Therefore, it is meaningful to develop a reliable infrastructure in which end-users or data creators are able to secure the ownership of data while being able to contribute to machine learning tasks in a privacy-preserving manner with reasonable financial incentivation. To achieve this goal, in this paper we develop a decentralized and secure computing infrastructure that enables an effective and privacy-preserving collaboration between the available end-users and data creators that are called computing nodes in this paper. These computing nodes can have restricted computing power and limited computing intelligence. Additionally, they can be scattered and untrustworthy to each other.

In recent years, several techniques have been proposed to achieve decentralized and privacy-preserving computing. In [6], Shokri *et al.* proposed a privacy-preserving deep-learning mechanism with secure multi-party computations to update the single initial deep learning model. However, the deep learning model is centralized while multiple parties contribute to the model training in a manner that guaranties the privacy-preservation. Federated learning introduced in [7]–[10] is a distributed machine learning method that enables model training on decentralized data. In this method, multiple copies of the central model can be available in distributed computing devices for training, which eliminates the single point of failure. However, both of these two methods must be managed by one centralized authoritative controlling agent, which may raise security and privacy concerns. To address these issues, in our work we develop a decentralized and secure computing paradigm, which does not have any centralized authoritative controlling agents, by exploiting blockchain, machine learning, and homomorphic encryption technologies. Blockchain is an emerging technology, which can be considered as an immutable and decentralized digital ledger [11], [12]. A blockchain is a growing list of records, called blocks, which are linked via cryptography. Each block contains the hash value of the previous block, the transaction data, and the timestamp. Due to the inherent cryptographic chaining, if a malicious party tries to manipulate certain transaction data, it will causes the changes of the hash values of the block containing the transaction and those of all the subsequent blocks, which can be easily detected. Therefore, generally speaking, blockchain technology provides a very promising solution for integrity security. Amongst various existing blockchain platforms, Ethereum and Bitcoin are two of the most widely adopted ones [11]–[13]. Compared with Bitcoin, Ethereum platform provides a trustful automation of programs via smart contracts that run on virtual machines. In our work, we exploit Ethereum blockchain to execute

the secure and privacy-preserving decentralized computing functionalities automatically.

Furthermore, our computing paradigm enables the effective decentralized and cooperative learning via an effective learning-model fusion mechanism. Fusing multiple learning models is an active area of research and fusion strategies in literature can be divided into two main categories: (1) *Late fusion* that comprise predicting the labels based on the labels given by each learning model to be fused and (2) *Early fusion* that takes the feature vectors given by the individual learning models as the inputs and learns a classifier on top of them. Although *late fusion* requires lower computational cost compared with *early fusion* in many practical applications as stated in [14]–[16], *early fusion* can achieve a more optimal way to combine learned models compared with *late fusion* [17]. In this work, our learning-model fusion mechanism belongs to the type of *early fusion*, in which the feature vectors to be fused present features with the highest level of abstraction. Additionally, we consider two strategies for designing the fusion mechanism: (1) using a fully connected structure with a single hidden-layer to map concatenated features to labels and (2) implementing gradual fusion to explore the uniqueness of the individual learning models and the correlation amongst the learning models.

To further enhance the security and achieve privacy-preservation, we design a encryption interface with a zero-knowledge proof protocol by exploiting homomorphic encryption (HE), which enables evaluating the performance of the contributed learning models without revealing the sensitive details of the learning models. HE technology is one form of encryption that allows the computation operations to be directly implemented in cipherspace and achieves an encrypted results that, when decrypted, match the results of the operations as if they had been performed on the plainspace [18]. The existing HE technologies can be generally classified into three main groups: (1) fully HE schemes, (2) partially HE schemes, and (3) somewhat HE schemes [19]–[25]. Considering the fact that somewhat HE schemes support more operations compared with partially HE schemes and require less computation power compared with fully HE schemes, we exploit Integer-Vector HE scheme [24], [25], which is a somewhat HE scheme, to develop the encryption interface in our computing paradigm. The authors would like to claim that the technologies presented in this paper has been include in a provisional patent [26].

The next section describes the problem setting for our work. Section III describes our proposed blockchain-powered decentralized and secure computing paradigm mechanism followed by the details of the implementation in Section IV. Simulation results and the conclusions are shown in Sections V and VI, repectively.

II. PROBLEM SETTING

Two of the main factors for the thriving of machine learning are: (1) the availability of sizable data-sets with generalized distributions, commonly known as big data, and (2) the availability of the computational power to process this big data, mainly in the form of large-scale GPU clusters.

Because of this, most profitable parties in the field of machine learning are large organizations, which hold both valuable big data and sufficient computational power to process it. As illustrated in Fig. 1(a), these large organizations collect data from data contributors to advance the capabilities of their machine learning techniques. One of the essential challenges for creating large-scale datasets via data acquisition, from multiple parties, is the issue of privacy and the related concerns. Potential data providers may not get motivated to share the data because of the high potential for data privacy violations. Additionally, collecting tremendous raw data from multiple parties results in a huge demand on communication bandwidth and a dramatically increased attack surface. Furthermore, a large amount of computational power is required by the central server to process the collected big data.

One solution is the implementation of distributed learning architectures [6]–[10]. As shown in Fig. 1(b), in distributed learning, rather than collecting and processing data in a single central server, data processing is distributed partially to the individual data providers. By doing so, the distributed learning is implemented in such a way, that the computing contributors process their local data by training the given machine learning models or their own machine learning models and then share the trained model with a central controlling agent. Since the data are not shared, we can say that the data privacy is preserved in this architecture. Additionally, the machine learning models are trained in distributed locations with smaller sets of data, and thus the computational power required by the individual computing contributors is much lower, compared with that of a central server. However, in this solution, the machine learning architecture is fully controlled by a authoritative agent in a centralized manner. It relies on the central authority to coordinate the activities of each entity in the system. Therefore, it is required that the computing contributors trust the central controlling agent, which may raise security and privacy concerns.

To mitigate this, we improve the distributed machine learning architecture presented in Fig. 1(b) and achieve the decentralized and cooperative machine learning architecture shown in Fig. 1(c). In a decentralized system, each entity is completely autonomous and responsible for its own individual behavior. In this architecture, the untrustworthy computing contributors have full control on their own deep learning models and private data. Additionally, the individual contributors are able to participate or leave the computing architecture, without disturbing the functionality and efficiency of the overall learning process. Also, the participation of the computing contributors is motivated by financial compensation that they will receive according to the value of their contribution. To achieve these objectives, we exploit the Ethereum blockchain and design the smart contract to secure the peer-to-peer transactions between the multiple untrustworthy parties to enable the autonomous decentralized and cooperative deep learning.
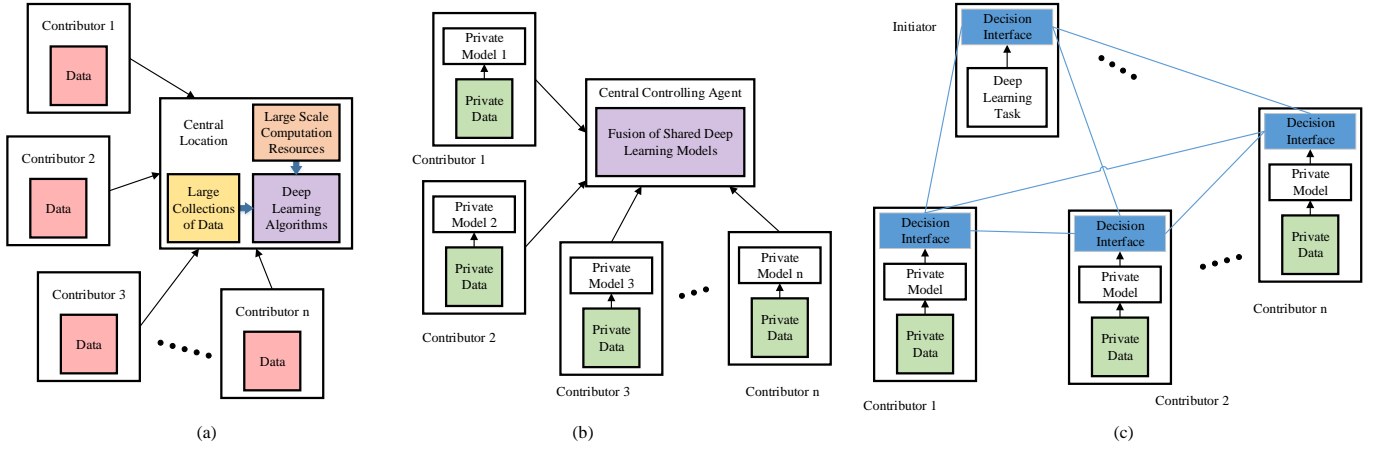
Fig. 1. (a) Centralized machine learning architecture where data are collected to centralized server with high processing and storage capability; (b) Distributed machine learning architecture where partial of training is distributed to the data contributors and the training process is fully controlled by a central controlling agent; (c) Autonomous cooperative and decentralized machine learning architecture with no central agents facilitated by blockchain service infrastructure.
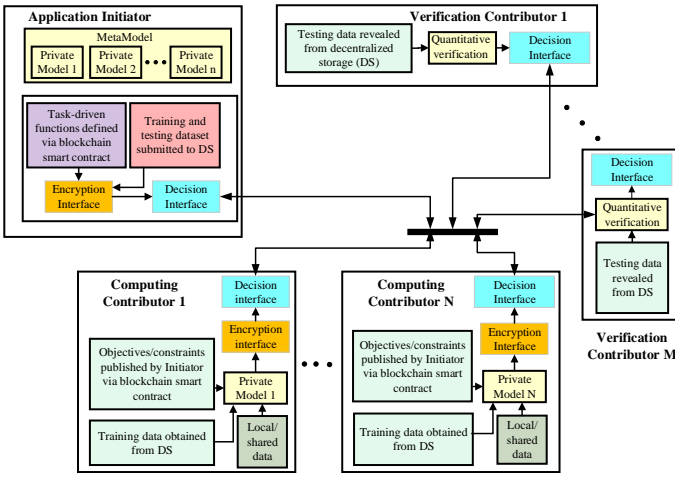

Fig. 2. Overview of our blockchain-powered decentralized and secure computing paradigm.

III. Proposed Blockchain-Empowered Cooperative Machine Learning Platform

The overview of our proposed blockchain-powered decentralized and decentralized computing paradigm is illustrated in Fig. 2. As shown in Fig. 2, our proposed mechanism is designed to enable the effective cooperation between the available and possibly scattered computing nodes to accomplish data-driven task that may require high computing power and intelligent and large dataset. The individual computing nodes can participate the computing paradigm by playing one of the three roles: (1) application initiators, (2) computing contributors, or (3) verification contributors. As detailed in Fig. 3(a), if the computing node act as an application initiator, it announces the data-driven applications and is responsible for defining the computing tasks, such as the objectives, constraints, the suggestions on the computing model structure, and the financial compensation commitments, via blockchain smart contract. The application initiators also provide the

verification contributors with a sample set of data to evaluate the performance of the learning models contributed by the computing contributors. If it is necessary, the application contributors also provide the computing contributors, which have computing power and computing intelligence, with the dataset to conduct the local training. The sharing of the dataset is realized via the decentralized storage (DS) such as The Interplanetary File System (IPFS) [27]. As shown in Fig. 3(b), the computing contributors train the machine-learning models locally for a given data-driven task by using a certain local data asset or the data shared by the associated application initiator. After training the local learning model successfully according to the criteria defined by the application initiator via smart contract, such as the accuracy is above 90 %, the computing contributors announce the completeness of the training via the blockchain platform and share the achieved the machine-learning model to the randomly selected verification contributors via the DS such as IPFS. The available verification contributors are passively and randomly selected to provide the hardware resources and verify the contributions of the locally trained learning models, which are claimed by the computing contributors, in a random and decentralized manner. As illustrated in Fig. 3(c), the verification is conducted according to the criteria defined by the application initiator via smart contract, such as whether the accuracy can be improved after fusing the claimed model. The majority voting amongst the verification contributors is used to determine the contribution of the corresponding computing contributors. The application initiator is informed about the majority voting conclusion. If this conclusion is positive, the transaction of the verified locally-trained machine-learning model, also called private model, is established between the application initiator and the associated computing contributor, in which the computing contributor receives the financial compensation and the application initiator obtains the access to the learning model in IPFS. Additionally, the verification contributors involved in the task also get compensated from initiators for their effort. After the time window assigned to the data-driven task ends,
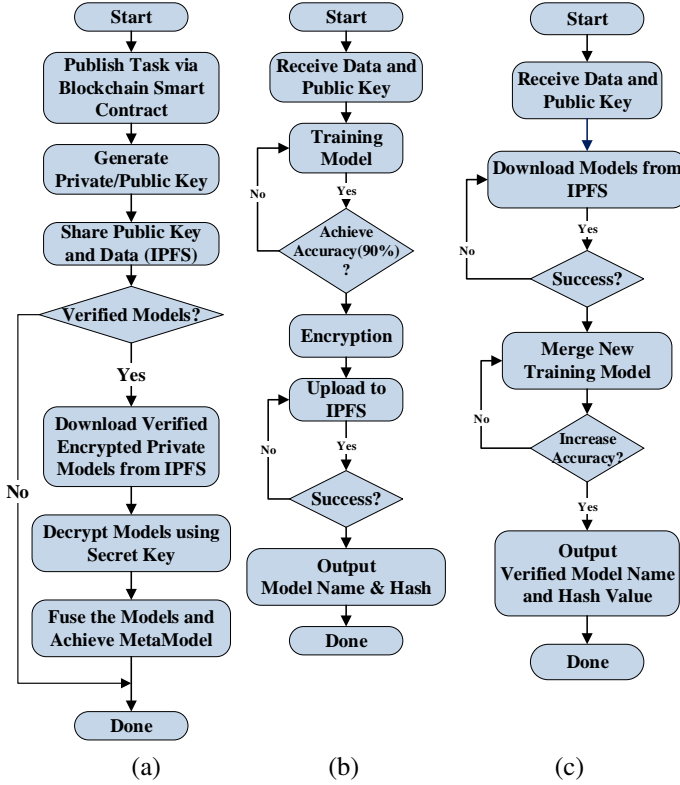
Fig. 3. The schematic diagram showing workflows for (a) application initiators, (b) computing contributors, and (c) verification contributors.

the application initiator fuses all the received verified private models to achieve the MetaModel that will be applied to address the data-driven task. The architecture of our proposed computing paradigm is considered asynchronous and adaptive since the computing and verification contributors can leave or join the task at their convenience.

IV. IMPLEMENTATION OF PROPOSED COMPUTING PARADIGM

In this section, we described the implementation of our proposed blockchain-powered decentralized and secure computing paradigm in details. The implementation of our proposed computing paradigm comprises of three layers: (1) application layer to conduct decentralized and secure computing, also called computing layer, (2) blockchain middleware to enable the secure peer-to-peer communications between the computing nodes including the transactions of the private models and the data sharing, and (3) software-defined networking (SDN)-enabled networking layer.

A. Decentralized Storage System

As shown in Fig. 2, decentralized storage (DS) system is one essential component of the computing layer to secure the sharing of data and private machine-learning models. DS system makes the access these data and learning models more affordable, faster and safer. Firstly, expensive centralized data servers are no longer needed in a DS system because every peer can cache the data required by other peers. Secondly,

optimized downloading from multiple local peers provides higher traffic throughput than that from a remote centralized data server. In addition, as shown in Fig. 3, after the data or private models are stored in the DS system, the unique hashes characterizing the fingerprint of these cyber assets are generated and shared, which ensures the integrity and the authorization of the sharing process.

B. Blockchain Middleware

Ethereum Blockchain-based middleware is designed to automatically control, manage and secure the system processes. First of all, consensus protocol, such as Proof-of-Work (PoW), Proof-of-Stake (PoS), and Proof-of-Authority (PoA) of blockchain provides an unsupervised secure environment where the authoritative agents are not necessary any more for the decentralized computing. Additionally, the distributed database of blockchain provides a shared, irremovable ledge of any events happened in time order on the system. It is convenient to trace the ins and outs of a event on blockchain ledge. Furthermore, blockchain smart contract enables the automation of system processes including the training, verification, transaction, and fusion processes of the decentralized and cooperative learning.

C. Homomorphic Encryption Interface

As shown in Fig. 4, the encryption interface is designed to enhance the security and enable the privacy-preservation of our computing paradigm. In this work, the encryption interface is developed by exploiting Integer Vector Homomorphic Encryption (HE) scheme. Figure 4 illustrates the overall mechanism of the encryption interface, which mainly consist of eight steps. Step 1: Encryption interface client of the application initiator generates the public key M and the secret key S' according to Integer Vector HE scheme, which is illustrated in Fig. 3(a). Step 2: The generated public key M is shared amongst the active computing and verification contributors, which is illustrated in Figs. 3(a)-(c). Steps 3 and 4: Computing contributors apply the received public key M to encrypt the machine learning models achieved locally and share the encrypted private models with the passively selected verification contributors for quantitative verification, which is illustrated in Fig. 3(b). Step 5: After receiving the encrypted private model, the verification contributors verify the performance of the private models by conducting the quantitative verification in cipherspace with the public key M. In the quantitative verification, the verification contributor fuses the received private model with the existing learning model in cipherspace and concludes that the private model is valuable if the overall accuracy increases after model fusion. Step 6: The majority voting amongst all the associated verification contributors is used to determine the contribution of the private model. The application initiator is informed about the majority voting conclusion. Step 7: If the majority conclusion is positive, the transaction of the encrypted private model between the associated computing contributor and the application initiator is established. At last, the application initiator decrypts the shared model with the secret key S'.
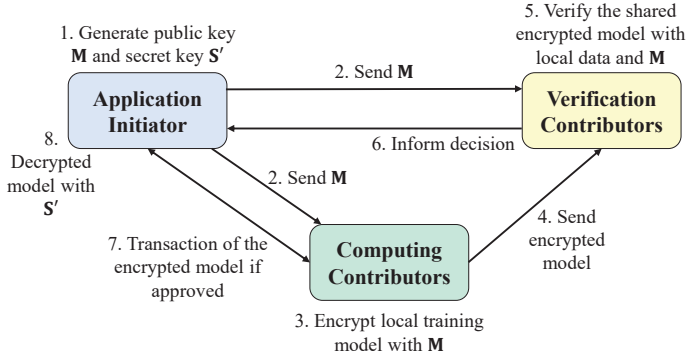
Fig. 4. The illustration of the mechanism of our encryption interface.

1) Integer-Vector Homomorphic Encryption: In our work, the integer-vector homomorphic encryption scheme, which supports three fundamental operations: addition, linear transformation, and weighted inner products, is exploited to develop the encryption interface. Letting $x \in \mathbb{Z}^m$ be the plaintext vector, $\mathbf{S} \in \mathbb{Z}^{\mathbf{m} \times \mathbf{n}}$ be the secret key, and $c \in \mathbb{Z}^n$ be the ciphertext vector, the encryption can be formulated as:

$$Sc = wx + e \qquad (1)$$

where e is a randomization term introduced to enable the encryption, which have elements smaller than w, and w is a large integer that controls the appropriate ratio between the plaintext and the introduced randomization.

Given the secret key S, the decryption can be performed as:

$$x = \lceil \frac{Sc}{w} \rfloor \qquad (2)$$

In this homomorphic encryption scheme, *key switching* method was proposed to convert the ciphertext in one cipherspace to another without decryption. This convert is realized via a public key $\mathbf{M}$ that is calculated as follows:

$$\mathbf{M} = \begin{bmatrix} \mathbf{S}^* - \mathbf{TA} + \mathbf{E} \\ \mathbf{A} \end{bmatrix} \qquad (3)$$

where $\mathbf{S}$ and $\mathbf{c}$ are the original secret key and ciphertext, respectively, $\mathbf{S}^*$ is an intermediate key satisfying $\mathbf{S}^* \mathbf{c}^* = \mathbf{Sc}$, $\mathbf{c}^*$ is the representation of $\mathbf{c}$ with each digit of $\mathbf{c}$, c_i represented as a l-bit binary number, and $\mathbf{A}$ and $\mathbf{E}$ are random and bounded matrices, respectively. A scalar l is selected to be large enough such that $|\mathbf{c}| < 2^l$, which determines the maximum value of $\mathbf{c}$ represented with l bits. Let $\mathbf{b_i}$ be the bit representation of the value c_i, we can obtain $\mathbf{c}^*$ as follows,

$$\mathbf{c}^* = [\mathbf{b_i}, \mathbf{b_{i+1}}, ..., \mathbf{b_n}] \qquad (4)$$

where n is the length of vector $\mathbf{c}$. Similarly, $\mathbf{S}^*$ can be obtained as follows:

$$\mathbf{B}_{ij} = [2^{l-1} S_{ij}, ..., 2S_{ij}, S_{ij}] \qquad (5)$$

where $\mathbf{B_{ij}}$ is the sub-vector of $\mathbf{S}^*$ that corresponds to the element S_{ij}.

Additionally, since the initial secret key $\mathbf{S}$ is a identity matrix of the dimension $n \times n$, the original ciphertext $\mathbf{c}$ is actually the original plaintext $\mathbf{x}$ itself. Let $\mathbf{S}' = [\mathbf{I}, \mathbf{T}]$, where $\mathbf{I}$ is identity matrix and $\mathbf{T}$, is a desired secret vector. By using the public key $\mathbf{M}$ defined in Eq. (3), the ciphertext corresponding to the desired secret key $\mathbf{S}'$ can be calculated as:

$$\mathbf{c}' = \mathbf{Mc}^* \qquad (6)$$

where $\mathbf{M}$ is a $(n+1) \times nl$ dimension matrix. Therefore, the resulting ciphertext $\mathbf{c}'$ is a integer vector with length $n+1$.

2) Implementation of Artificial Neural Networks-based Machine Learning Model in Cipherspace: As shown in Fig. 4, the essential component of our homomorphic encryption is to implement the artificial neural network (ANN)-based machine learning model in cipherspace by using integer-vector homomorphic encryption (IVHE) scheme. ANN implementation mainly comprises of summation, vector addition, vector multiplication with scalar, vector dot product, pooling operations, convolution operations, and nonlinear activation functions. Most of these operations are supported by the IVHE scheme, except pooling operations, convolution operations, and nonlinear activation function realizations. To enable the implementation of pooling operations in cipherspace, we currently assume the computing contributors adopt the average pooling or summation pooling for training their machine learning model locally. Under this assumption, the pool operations can be realized via the summation followed by a division by a integer, which is supported by the IVHE. The convolution operations can be implemented in cipherspace by calculating the vector multiplication operations. Additionally, we mainly consider two types of activation functions in the current work: sigmoid function, $\sigma(x) = 1/(1 + e^{-x})$, and ReLU function, $ReLU(x) = max(0, x)$. To implement the sigmoid functions in cipherspace, we leverage the Taylor series expansion to achieve the k-th order polynomial approximation of the sigmoid function. For example, if $k = 3$, the polynomial approximation is $\bar{\sigma} = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{348}$ that is supported by the IVHE scheme. To enable the implementation of ReLU function, which is a piecewise linear operation with discontinuity at $x = 0$, in cipherspace, we currently constrain the secret and public keys to contain the non-negative elements only. By doing so, there is no sign changes while encrypting. The ReLU function is executed in cipherspace via the random dropout.

There still remain two challenges of implementing the ANN-based machine learning model in cipherspace. First, most weight values and input data for ANNs are floating-point values that cannot directly be supported by our adopted homomorphic encryption scheme. Additionally, implementing the average pooling and calculating the polynomial approximation of sigmoid function require the multiplications with floating-point numbers, which is also not supported by our adopted homomorphic encryption scheme. To address this issue, we introduce predetermined and unified scaling factors to convert the floating-poitn values to integers. A final propagated scaling factor is used to scale down the output values of the ANNs to the original values. Second, as discussed in Section IV-C1, inter-vector homomorphic encryption increases the length of ciphertext vector by 1 compared with that of the plaintext vector. This difference between the dimensions of the ciphertext and plaintext raises challenges in implementing the ANN

operations that requires consistency in dimensions such as feed-forward operations. To address this issue, we develop two encryption strategies as follows:

a) Element-wise Encryption Strategy: The essential idea of our element-wise encryption strategy is to encrypt the matrices and vectors in a manner of element by element. By doing so, the additional components introduced by the homomorphic encryption can be addressed in the third dimension, which ensures the consistency on the original dimensions. To illustrate our strategy, we use a fully-connected neural network (NN) as an example shown in Fig. 5(a). The details of the implementation of the fully-connected NN in cipherspace are illustrated in Fig. 5(b).

As shown in Fig 5(a), the fully-connected ANN has an input vectors with length of n, a weight matrix, Weight Matrix 1, with the dimension of $n \times m$ resulting in a hidden layer of size m adopting sigmoid function as the activation function, a weight matrix, Weight Matrix 2, with dimension $m \times k$, and a output vector of length k. To enable the implantation of our element-wise encryption strategy, Weight Matrices 1 and 2 are represented with dimensions $n \times m \times 1$ and $n \times k \times 1$, respectively, and multiplied with a scaling factor p to ensure all the matrix elements to be integer. Another scaling factor q is introduced to convert the input elements to be integer. Considering the fact that, if Weight Matrix is encrypted, the following structure of the fully-connected NN is meaningless for a malicious party, it is reasonable to focus on encrypting Weight Matrix 1 only to achieve a tradeoff between the high security and low computational complexity. As shown in Figs. 5(a) and (b), we consider each element of Weight Matrix 1 as a vector with length 1 in the 3rd dimension, and apply the element-wise encryption strategy, which results in a weight matrix with the dimension of $n \times m \times 2$ in cipherspace. The dot product operation between the scaled and encrypted Weight Matrix 1 and the scaled input vectors is executed by using IVHE, which results in a pair of weighted inputs in cipherspace. By achieving a polynomial approximation of the sigmoid activation function, the hidden layer with the approximated sigmoid functions is encrypted via the homomorphic encryption resulting in a hidden layer pair in cipherspace. For each of the encrypted hidden layer in the pair, a dot product operation is performed with the scaled Weight Matrix 2, which results in an output pair in cipherspace. Additionally, as shown in Fig 5(b), a scaling-down operation is required during the decryption conducted in the encryption interface in application initiator.

Figure 5(c) illustrates the implementation of our element-wise encryption strategy to execute a convolutional neural network (CNN). Similar to the previous example, the convolution filter is converted to a pair of scaled and encrypted filters having the dimension of $m \times r$. A dot product operation is performed on each encrypted convolution filter of the pair with the scaled input vector. Then a pair of hidden layers in cipherspace is achieved by implementing ReLU operations, sum-pooling, and reshaping. Another dot product operation is executed on each of the hidden layer in the pair with the scaled Weight Matrix 2, which results in an output pair in cipherspace. Since, in this example structure, the encrypted

ReLU activation function is realized via random dropout and the operations of sum-pooling and reshaping are executed in cipherspace using IVHE scheme, no additional scaling is introduced through these operations. Therefore, the final output scaling factor remains lower compared to the previous example fully-connected NN using sigmoid activation function.

b) Matrix-Pair-wise Encryption Strategy: Our matrix-pair-wise encryption strategy is performed on a pair of neighboring weight matrices. To illustrate our strategy, we use a CNN as an example shown in Fig. 6. As illustrated in Fig. 6, this CNN has two convolution layers having the dimensions of $m \times 1 \times r$ and $m \times r \times l$, respectively, where r is the number of convolution filters in the first convolutional layer and l is the number of convolution filters in the second convolutional layer.

In our encryption strategy, the dimension of encryption is carefully selected such that there is no dimension mismatch while executing the CNN-based machine learning model in cipherspace. Additionally, the first convolution layer is encrypted via IVHE scheme by leveraging the 3rd dimension, which results in a scaled and encrypted convolutional filter with dimension $m \times 1 \times (r + 1)$. Similarly, the second convolution layer is encrypted on the 2nd dimension, which results in a scaled and encrypted convolutional filter with dimension $m \times (r + 1) \times l$. By doing so, the convolution operation can be performed as a pair without any dimension mismatch.

D. Learning-Model Fusion Mechanism

As illustrated in Fig. 2, the success of our decentralized and secure computing paradigm requires an efficient learning-model fusion mechanism, with which the application initiator is able to integrate the verified private models and achieve an effective MetaModal for the targeted data-driven task. For designing the learning-model fusion mechanism, it is necessary to treat the computing models, provided by the computing contributors, as separate entities, to ensure the fused structure is dynamic.

Figure 7 illustrates the structure of our proposed fusion mechanism. In our mechanism, the upper-layer feature vectors $\mathbf{f_i}$ from individual verified private models are concatenated to form a concatenated feature layer $\mathbf{f_c}$. As shown in 7, the upper-layer feature vector $\mathbf{f_i}$ can be the input of the output layer for the private model i. One fully-connected neural network (NN) is designed to fuse the features characterized by the individual private models. This fully-connected NN uses the concatenated feature layer $\mathbf{f_c}$ as its input layer and has a hidden layer $\mathbf{h}$ with the length of $\sum_{i=1}^{n} | l_i |$, where $| l_i |$ is the number of the labeled classes in the ith private model. TO design the fully-connected NN, it is essential to design its weight matrices $\mathbf{A}$ and $\mathbf{B}$. Currently, we consider two strategies for learning the optimum values for weight matrices $\mathbf{A}$ and $\mathbf{B}$ and designing the learning-model fusion mechanism.

1) Strategy I: In this strategy, the weight matrices $\mathbf{A}$ and $\mathbf{B}$ are initialized randomly without any additional constraints and are optimized via the backpropagation algorithm [28] in which
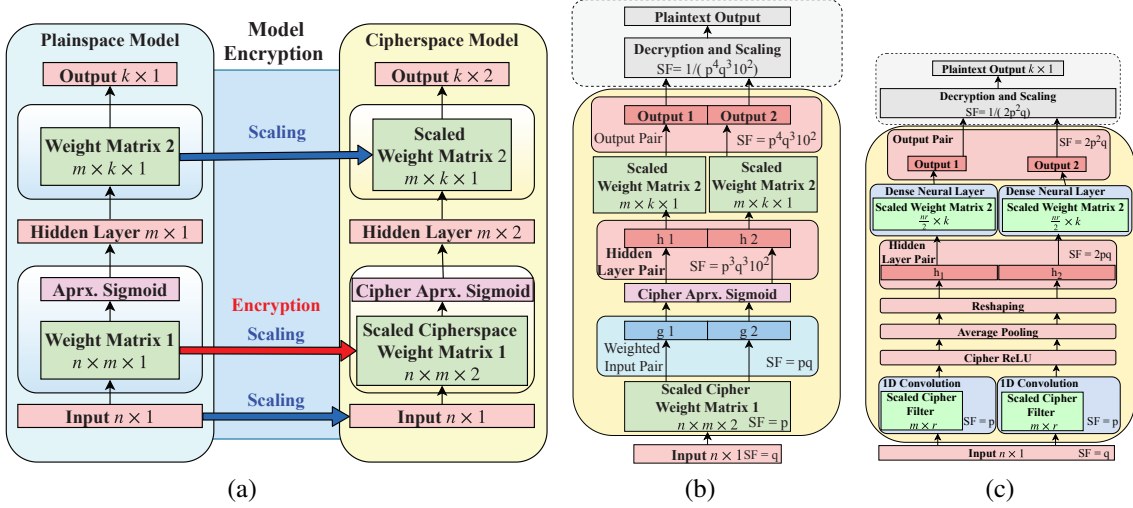
Fig. 5. The illustration of our element-wise encryption strategy: (a) using a fully-connected NN as an example, (b) detailing the execution of the fully-connected NN in cipherspace, and (c) using a CNN as an example.
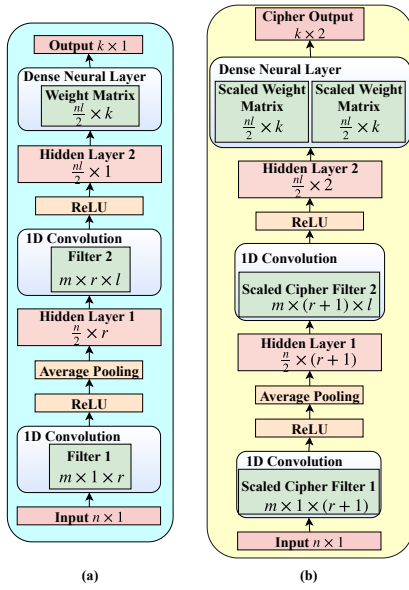

Fig. 6. Illustration of our matrix-pair-wise encryption strategy: (a) a CNN-based machine learning model in plainspace and (b) the corresponding CNN-based machine learning model in ciperspace.

the ith element in the hidden layer $\mathbf{h}$ and the jth element in the output layer $\mathbf{y}$ are calculated as follows:

$$h_j = \sum_{i=1}^{|\mathbf{fc}|} A_{ij}^T \cdot fc_i \tag{7}$$

$$y_j = \frac{exp(\sum_{i=1}^{|\mathbf{h}|} B_{ij}^T \cdot h_i)}{\sum_{k=1}^{d} exp(\sum_{i=1}^{|\mathbf{h}|} B_{ik}^T \cdot h_i)} \tag{8}$$

2) Strategy II: This strategy is developed to achieve to goals: (1)learning the uniqueness of the features characterized by the individual verified private models, and (2) exploring the correlation amongst the features presented by the individual private models. To achieve this goal, a gradual fusion is designed, in which the weight matrix $\mathbf{A}$ is initialized as a

concatenated matrix formulated in Eq. (9) and the weight matrix $\mathbf{B}$ is initialized with the concatenation of the identity matrices, as formulated in Eq. (10).

$$\mathbf{A}_{init} = \begin{bmatrix} \mathbf{W_1} & 0 & 0 & \dots & 0 \\ 0 & \mathbf{W_2} & 0 & \dots & 0 \\ 0 & 0 & \mathbf{W_3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \mathbf{W_n} \end{bmatrix} \tag{9}$$

where a diagonal weight matrix $\mathbf{W_i}$, which has the dimension of $(|\mathbf{f}_i|, |\mathbf{l}_i|)$, is initialized randomly.

$$\mathbf{B}_{init} = \begin{bmatrix} w_{111}=1 & 0 & 0 & \dots & 0 \\ 0 & w_{122}=1 & 0 & \dots & 0 \\ 0 & 0 & w_{133}=1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & w_{1dd}=1 \\ w_{211}=1 & 0 & 0 & \dots & 0 \\ 0 & w_{222}=1 & 0 & \dots & 0 \\ 0 & 0 & w_{233}=1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & w_{2dd}=1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n11}=1 & 0 & 0 & \dots & 0 \\ 0 & w_{n22}=1 & 0 & \dots & 0 \\ 0 & 0 & w_{n33}=1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & w_{ndd}=1 \end{bmatrix} \tag{10}$$

where n is the number of the verified private models to be fused and d denotes the number of class labels.

The elements of the weight matrices $\mathbf{A}$ and $\mathbf{B}$ are optimized by using our gradual fusion method that consists of two stages. In the initial stage, only the diagonal non-zeros weights of matrix $\mathbf{A}$ by using backpropagation algorithm, which targets at learning the uniqueness of the features characterized by the individual private models. In the second stage, all of the weights in $\mathbf{A}$ are updated by using backpropagation algorithm, which targets at exploring the correlations between the features
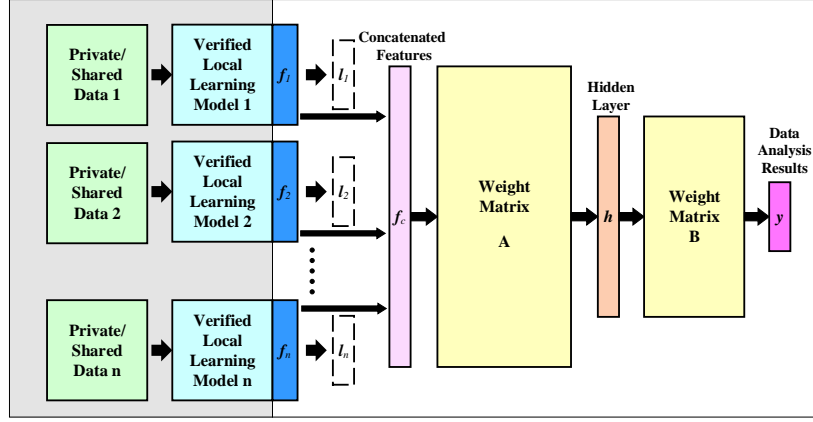
Fig. 7. Illustration of our proposed learning-model fusion mechanism.

characterized by the individual private models. Accordingly, the ith element in the hidden layer $\mathbf{h}$ and the jth element in the output layer $\mathbf{y}$ are calculated as follows:

$$h_j = \sum_{i=1, A_{ij} \in \bar{\mathbf{W}}}^{|\mathbf{fc}|} A_{ij}^T \cdot fc_i + \gamma \sum_{i=1, A_{ij} \notin \bar{\mathbf{W}}}^{|\mathbf{fc}|} A_{ij}^T \cdot fc_i \quad (11)$$

where the parameter γ is set as 0 in the intial stage and as 1 in the final stage, $\bar{\mathbf{W}} = \mathbf{W_P} \leftarrow \sum_{k=1}^{p-1} | \mathbf{l_k} | < i \leq \sum_{k=1}^{p} | \mathbf{l_k} |$.

$$y_j = \frac{exp(\sum_{i=1, B_{ij}=w_{jpp}}^{|\mathbf{h}|} B_{ij}^T \cdot h_i + \gamma \sum_{i=1, B_{ij} \neq w_{jpp}}^{|\mathbf{h}|} B_{ij}^T \cdot h_i))}{\sum_{k=1}^{d} exp(\sum_{i=1, B_{ik}=w_{ipp}}^{|\mathbf{h}|} B_{ik}^T \cdot h_i + \gamma \sum_{i=1, B_{ik} \neq w_{ipp}}^{|\mathbf{h}|} B_{ik}^T \cdot h_i)} \quad (12)$$

where $p \in \{1, 2, 3 \ldots, d\}$, and d is the number of labeled classes.

V. SIMULATION RESULTS

In this section, the performance of our proposed blockchain-powered decentralized and secure computing paradigm is evaluated by considering three case studies. To achieve this goal, we develop a Blockchain-powered Software-defined networking (SDN)-based testbed as detailed in Section V-A.

A. Blockchain-Powered SDN-Based Testbed

Fig. 8 (a) shows a picture of our Blockchain-powered SDN-based testbed, in which the computing nodes, including one application initiators, three computing contributors, and three verification contributors, are simulated by using two popular embedded systems, Raspberry PI and NVIDIA Jetson. The communications and cooperative computing amongst the computing nodes are supported by the blockchain middleware and the SDN-based peer-to-peer networking layer. The integration of the computing layer, blockchain middleware, and the SDN-enabled networking layer in our computing testbed are illustrated in Fig. 9. Each computing node, which is simulated via Raspberry PI or NVIDIA Jetson, has (1) one or multiple blockchain clients to interact with the blockchain middleware, (2) multiple ethernet ports to interact with the SDN-enabled

networking layer, and (3) one or more decentralized storage (DS) clients to enable the decentralized storage systems.

In our testbed, the blockchain middleware is developed by exploiting a decentralized application (DApp) on the Ethereum platform in which the engine and language of smart contract were Ethereum Virtual Machine (EVM) and Solidity. The consensus protocol is set to be Proof-of-Authority (PoA). The block interval is set to 15 s and all sealers are initialized with a small number of tokens. Additionally, the smart contract in Ethereum is written by Solidity and the blockchain event listener client is leveraged to provide the interface of the logging facilities of the Ethereum virtual machine. Every computing node, which is willing to participate in a certain decentralized computing task via the blockchain smart contract, is required to send a transaction through Ethereum blockchain to contract with a considerate amount of security deposit of tokens. By doing so, the smart contract is able to record the identities of all the participants and forward the tasks such as training and verification to the participants appropriately. The deployment of smart contract is considered as a transaction as well, which provides the other nodes with the address necessary to access the smart contract. As shown in Fig 9, the DS system in our testbed is realized by using Interplanetary File System (IPFS) that utilizes the peer-to-peer network to share and store hypermedia on the Internet. Since the whole simulation was running on a private network, a shared swarm key was used to authorize reliable nodes to join IPFS. Furthermore, our computing layer is developed in Python because most frameworks of machine learning are realized there. Web3.py and ipfs-api.py provide access to Ethereum blockchain and IPFS clients to realize the transactions and file sharing, respectively. Tensorflow is a deep learning framework with Python bindings, which is adopted for the task of machine learning in our testbed.

To further demonstrate our testbed, the screenshot of the terminals for a computing initiator, a computing contributor, and a verification contributor during one experiment are shown in Figs. 8(b)-(d), respectively. As shown in Fig. 8(b), the application initiator's operation comprises three main processes that are identified with three red rectangular boxes: (1) being

initialized to publish one data-driven task, characterize the objectives and constraints of the task via smart contract, upload the data to be shared to IPFS, and share the public key generated via homomorphic encryption interface, (2) receiving the verified models that are concluded according to the majority votes of all the active verification contributors, and (3) implementing decryption and fusion on the received verified models. As shown in Fig. 8(c), the computing contributor's operation mainly consists of four processes: (1) participating the data-driven task, (2) terminating the training task when the accuracy meets the given/self-defined criterion, (3) encrypting the achieved private model via homomorphic encryption interface and uploading the encrypted private model to IPFS, and (4) announcing the encrypted private model for assessment. As shown in Fig. 8(d), the verification contributor's operation includes three main processes: (1) being passively and randomly selected and initialized for the given data-driven task, (2) receiving the all announced private models, and (3) verifying the received private models according to the criterion defined in smart contract and publishing the verification results.

B. Case Study I

In this case study, we focus on evaluating the performance of our proposed blockchain-powered decentralized and secure computing paradigm in a secure environment. In other words, the functionality of the homomorphic encryption interface is not considered. We assume one application initiator publishes a data-driven task on classifying the 10 image classes provided by the MNIST handwritten digit database [29] and three computing contributors, and three verification contributors, which are considered to be randomly selected, participate in the task. We consider that each computing contributor develops its own CNN-based machine learning model whose structure parameters are summarized in Table I. As shown in Table II, the local data available to each computing contributor only present a partial view of the whole dataset. The accuracy of the CNN models developed by the individual computing contributors for classifying their local verification data is shown in Fig. 10, all of which are above 90 %. In this scenario, the criterion of determining whether the local learning model is trained successfully is achieving the accuracy of 90 %. Therefore, all of the three private CNN models are encrypted and published to the three verification contributors for assessment. After receiving the verified private models according to the majority voting amongst the verification contributors, the application initiator fuses the models by using the two strategies introduced in Section IV-D to achieve the MetaModal. Assume that the application initiator decrypts and fuses the private models as soon as receiving them, the classification accuracy achieved by the MetaModal is shown in Fig. 11. From Fig. 11, it is clear that the classification accuracy increases as more private models are fused. This is reasonable since the individual private models are achieved using the local data that only characterize the partial features of the whole dataset. Furthermore, it can also see from Fig. 11 that Fusion Strategy II slightly outperforms the Fusion Strategy I when fusing multiple private models.

C. Case Study II

In this case study, we focus on evaluating the performance of homomorphic encryption interface in our proposed decentralized and secure computing paradigm by using the testbed shown in Figs. 8 (a) and 9. The local data that can be accessed by the individual computing contributors are detailed in Table III, which show that the individual computing contributors only have partial view of the entire dataset. Additionally, we assume that each computing contributor locally trains a CNN private learning model with a similar structure as illustrated in Table IV. To perform our homomorphic encryption interface, the floating-point input data and weight parameters of the CNN-based learning models are converted to be integer by introducing appropriate scaling factors p and q as shown in Fig 5. In this simulation, we evaluate the performance of the final MetaModal when the computing contributors select the scaling factors $p = 2^5$ or 2^7 and $q = 1000$. Furthermore, we assume that in the encryption interface of the computing contributors, the private CNN-based learning models are encrypted via the first convolution layers. In the encryption interface of the application initiators, the verified private models are decrypted and fused to achieve MetaModals via Fusion Strategy II. The accuracy of the MetaModal achieved with and without homomorphic encryption interface are shown in Table V, respectively. From Table V, we can see the MetaModal achieved by using the encryption interface with the scaling factor $p = 2^7$ outperforms that with the scaling factor $p = 2^5$ and achieves comparable accuracy as the original MetaModal. This is reasonable because that the errors caused by rounding the parameters of the CNN-based private models to integers increases when the scaling factor is lower.

D. Case Study III

In this case study, we compare our proposed computing paradigm to a widely recognized distributed machine learning paradigm, federated learning [30] in a data-driven task on classifying the 10 image classes provided by the MNIST handwritten digit database [29]. In federated learning, the local contributors initialize their model by referring the global initialized weights and then update their own models for a certain numbers of local epochs. The weights of the local models are averaged as the new global weights, which completes the first round of model update. The updated global weights are distributed to local computation contributors for a next round of model updating procedure. This process is called Federated Averaging (FedAvg).

To achieve a fair comparison, we assume the decentralized computing is conducted in a secure environment. Therefore, in our computing paradigm, the functionality of our homomorphic encryption interface is not considered and only one verification contributor is needed. For federated learning, we assume that the global average process ignores the random selection of local weights and collect all the weights achieved by the available local models. In addition, we assume that each computing contributor adopts a three-layer CNN model in both of the computing paradigms. In our computing paradigm, there are three computing contributors, each of which conducts up to
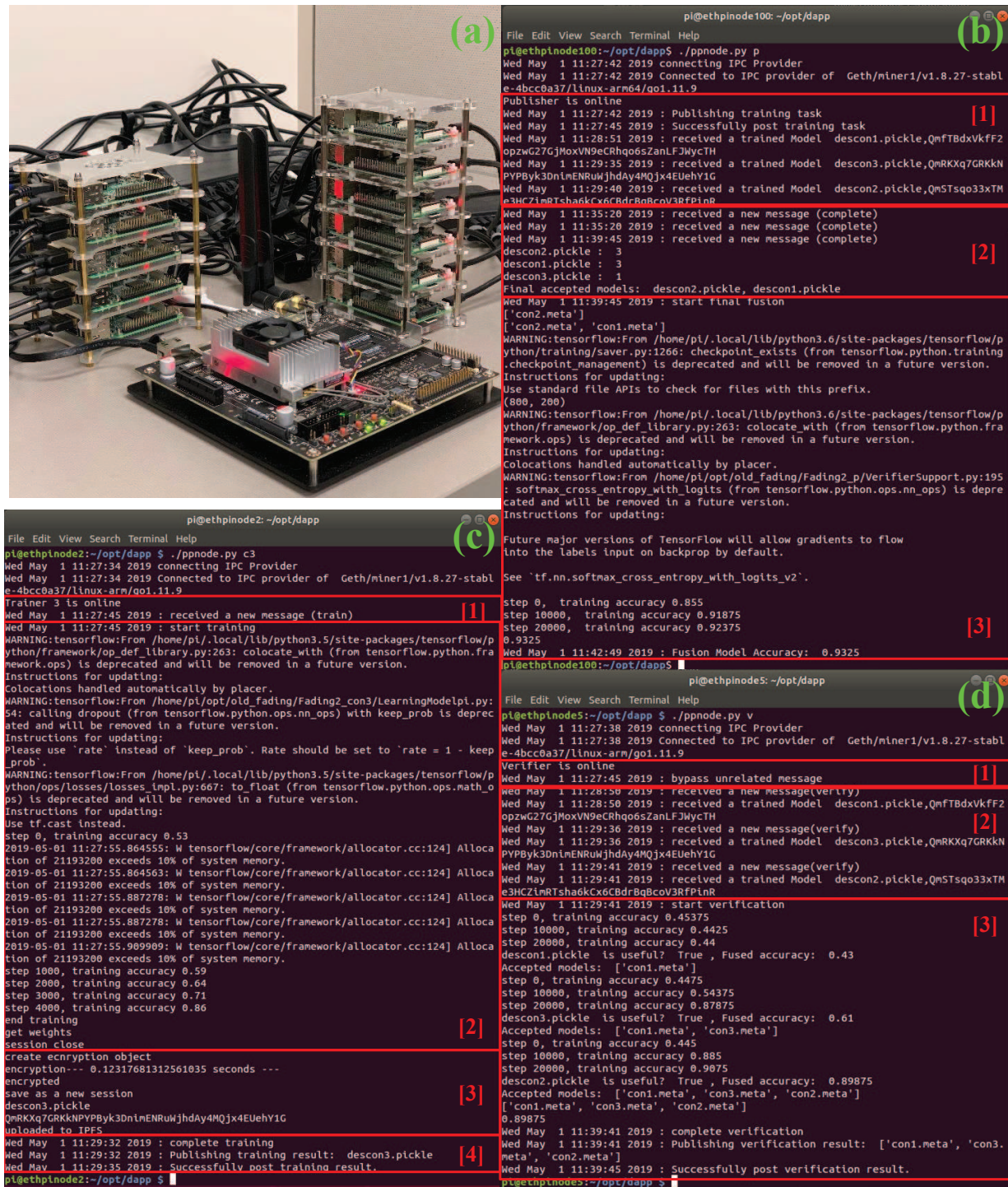
Fig. 8. (a) A picture of the testbed, and the terminals of (b) Application Initiator, (c) Computing Contributor, and (d) Verification Contributor.

100 training epoches locally, one verification contributor, and one application initiator. In federated learning, there are four local contributors training the local models, each of which involves in five rounds of model updating and conducts 20 training epoches locally in each round.

Furthermore, in this case study, we consider two scenarios. In the first scenario, we consider a diverse set of local training samples of the MNIST data available to the local contributors.

Each of these local training samples includes the data associated with all of the 10 image class labels, and the distribution of the local training samples are not identical amongst these contributors. In the second scenario, we consider that data are distributed amongst contributors in a way that only part of the 10 image classes are available to most of the contributors. The data distribution used in this scenario is similar to Table II.
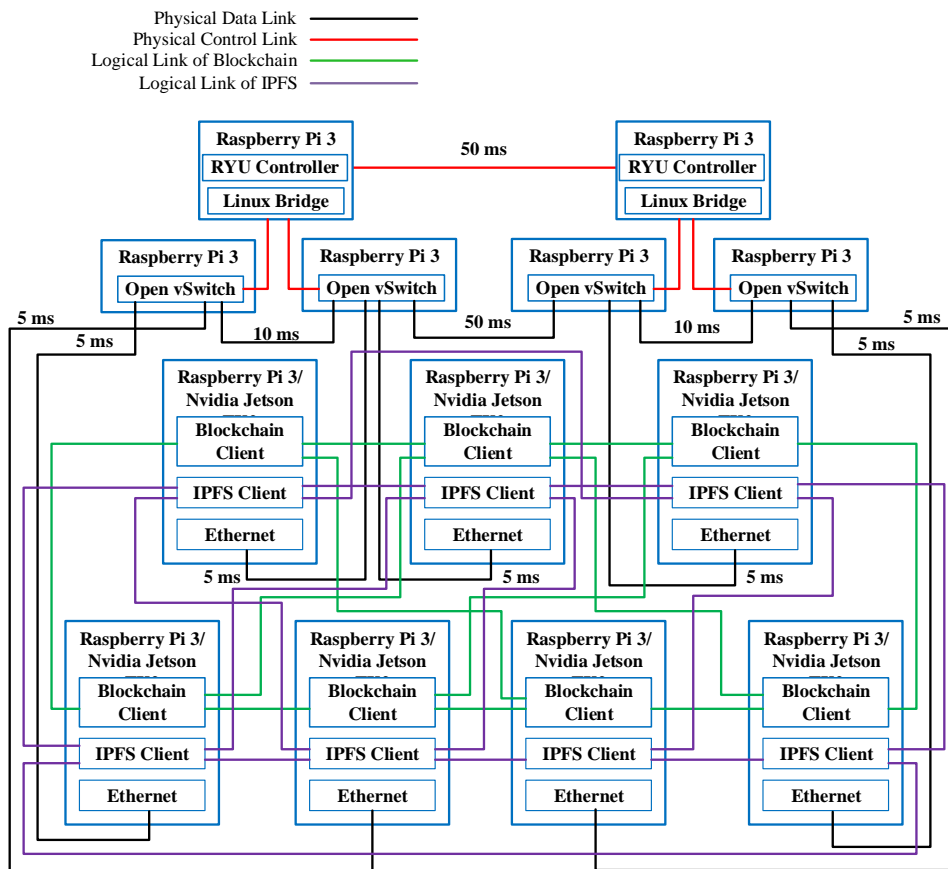
The comparison result is shown in Fig. 12, from which

Fig. 9. (a) Illustration of implementing our blockchain-powered and SDN-based decentralized and secure computing testbed.

TABLE I
PARAMETERS OF THE LOCAL LEARNING MODELS CONSIDERED IN CASE STUDY I

	Computing Contributor 1	Computing Contributor 2	Computing Contributor 3
	CNN parameters		
Inputs	28x28 images		
Convolution layer 1	32 5x5 kernels	64 5x5 kernels	32 5x5 kernels
Pooling layer 1	2x2 maximum pooling	2x2 maximum pooling	2x2 maximum pooling
Convolution layer 2	16 5x5 kernels	32 5x5 kernels	32 10x10 kernels
Pooling layer 2	2x2 maximum pooling	2x2 maximum pooling	2x2 maximum pooling
Convolution layer 3	8 2x2 kernels	16 2x2 kernels	16 4x4 kernels
Reshaped vector (Convolution layer 3 output are flatten as a vector of size)	7x7x8=392	7x7x16=784	7x7x16=784
hidden layer	Fully connected hidden layer with size 500		
Output	10 labels with softmax activation		
Training method	Adam Optimizer		
Batch size	50		
Learning rate	0.0001		
Maximum number of epochs	100		

it can be seen that both computing paradigms achieve a comparable classification accuracy. In the first scenario where all the contributors have a good view of the entire dataset, federated learning slightly outperforms our framework. In the second scenario, where most contributors only have partial views of the entire dataset, our computing solution is slightly better than the federated learning. Furthermore, our computing solution provides higher integrity security via removing the

authoritative controlling agent and introducing the verification contributors. The main functionality of verification contributors is for security purpose rather than for increasing the final accuracy, which also results in the relatively lower accuracy achieved by our solution in the first scenario shown in Fig. 12. Furthermore, the communication cost required by federated learning is higher than that required by our solution. This is caused by the centralized management and the repeated

TABLE II
SUMMARY OF THE DATASET AVAILABLE TO THE INDIVIDUAL
CONTRIBUTORS IN CASE STUDY I

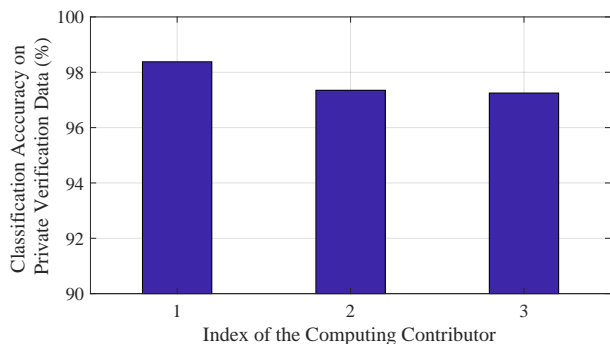| Contributor	Set of Labels	No. Training Data	No. Verification Data
Verifiers	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	1000	1000
Computing Contributor 1	{0, 1, 2, 3, 4}	1000	1000
Computing Contributor 2	{0, 6, 7, 8, 9}	1000	1000
Computing Contributor 3	{5, 6, 7, 8, 9}	1000	1000

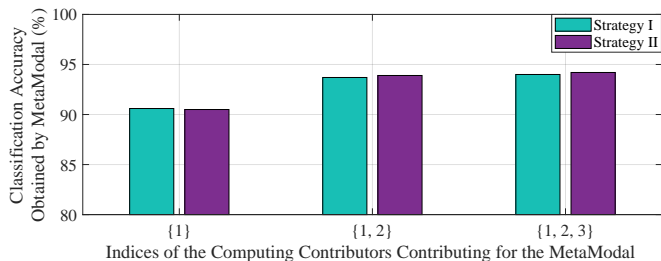Fig. 10. Accuracy obtained by each computing contributor by using their local verification data.

Fig. 11. Comparison of average classification accuracy obtained by Meta-Modal using strategies I and II versus the indices of the computing contributors contributing to the MetalModal.

collection of the local weights during the process FedAvg.

E. Case Study IV

In this case study, we evaluate the performance of our decentralized and secure computing paradigm with the functionality of the homomorphic encryption interface in a data-

TABLE III
SUMMARY OF THE LOCAL DATA AVAILABLE TO THE INDIVIDUAL
CONTRIBUTORS IN CASE STUDY II

Contributor	Set of Labels	No. Training Data	No. Verification Data
Computing Contributor 1	{0, 1, 2, 3, 4, 5, 6}	1000	1000
Computing Contributor 2	{0, 1, 2, 3, 4, 8, 9}	1000	1000
Computing Contributor 3	{0, 1, 2, 6, 7, 8, 9}	1000	1000

TABLE IV
PARAMETERS OF THE CNN-BASED PRIVATE MODELS IN CASE STUDY II

CNN parameters	
Inputs	28x28 images
Convolution layer 1	32 5x5 kernels
Pooling layer 1	2x2 average pooling
Convolution layer 2	16 5x5 kernels
Pooling layer 2	2x2 average pooling
Reshaped vector (Pooling layer 2 output are flatten as a vector of size)	7x7x16=784
Output	10 labels with softmax activation
Training method	Adam Optimizer
Batch size	50
Learning rate	0.0001
Maximum number of epochs	100

TABLE V
COMPARISON OF THE ACCURACY ACHIEVED BY METAMODALS IN
DIFFERENT SITUATIONS IN CASE STUDY II

Indices of Fused Models	Classification Accuracy for Fused Models (%)		
	Original	Encrypted $(p = 2^7)$	Encrypted $(p = 2^5)$
{1}	91.5	91.5	89.6
{1, 2}	94.8	94.7	93.0
{1, 2, 3}	95.8	95.8	93.8

driven task of detecting a fading channel by using link power data. The confidential data used in this task are from Link Power database provided by NASA Glenn Research Center. In the simulation, we assume that one application initiator publishes this data-driven task and there are three computing contributors and three randomly selected verification contributors participating in the task. The link power data are randomly divided into 4 sets each of which contains 10000 training data and 1000 testing data. Of these four sets, three sets are used as the local data, each of which is available to one of the three computing contributors. The other set is considered accessible by assigned to the three randomly selected verification contributors in the testbed. Additionally, we assume that each computing contributor participating in the task adopts a CNN model with 2 convolution layers and ReLU activation function. In this case study we consider both
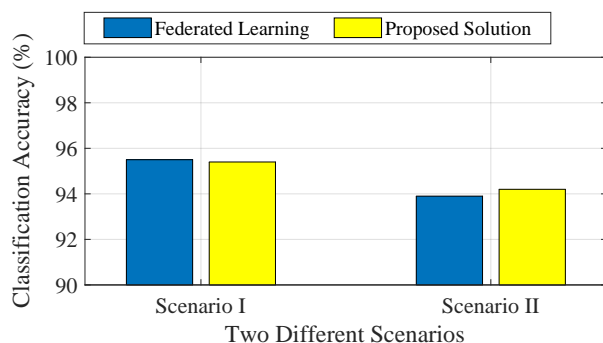

Fig. 12. Performance comparison between our computing paradigm and federated learning in two different scenarios

Element-wise and Matrix-Pair-wise homomorphic encryption strategies.

To demonstrate the operation of our homomorphic encryption interface, Table VI summarizes the accuracies obtained by each computing contributor with its private model and local testing data with and without the encryption interface. From these results, it is clear that implementing the encryption interface slighted reduces the classification accuracies, which is the cost to achieve higher security and privacy preservation. Additionally, Table VII shows the classification accuracy achieved by the MetaModal of the application initiator, which is obtained by fusing the individual verified private model via fusion strategy II. From the results in Table VII, which is obtained considering the functionality of homomorphic encryption interface, it is clear that although the final result is satisfactory the encryption interface slightly reduces the accuracy due to the introduction of randomization as explained in Section IV-C and rounding error as discussed in Section V-C.

Furthermore, we study the impact of the homomorphic encryption interface on the execution time. Tables VIII summarizes the overhead for encryption and the time consumed to achieve a local private model by each computing contributor with and without encryption interface. Homomorphic encryption interface is based on Element-wise Encryption. The results are obtained on a Corei5 CPU for 1000 testing data. ML models are executed on TensorFlow graphs. Encryptions and decryptions are performed using Python Numpy API. It can be seen that the execution time with encryption interface is only 2 times higher compared with that without encryption interface.

VI. CONCLUSIONS

Availability of computation power and data are two of the main reasons for the success of machine learning in a variety of application areas. However, both acquisition of processing power and data can be expensive. In many instances these challenges are addressed by relying on an outsourced cloud-computing vendor. However, although these commercial cloud vendors provide valuable platforms for data analytics, they can suffer from a lack of transparency, security, and privacy-preservation. Furthermore, reliance on cloud servers prevents applying big data analytics in environments where the computing power is scattered. Therefore, more effective computing paradigms are required to process private and/or scattered data in suitable decentralized ways for machine learning. To pave the way to achieve this goal, a decentralize, secure, and privacy-preserving computing paradigm is proposed in this paper to enable an asynchronized cooperative computing process amongst scattered and untrustworthy computing nodes that may have limited computing power and computing intelligence. This paradigm is designed by exploring blockchain, decentralized learning, homomorphic encryption, and software defined networking(SDN) techniques. The performance of the proposed paradigm is evaluated by considering different scenarios and comparing to a widely recognized distributed machine learning paradigm, federated learning, in the simulation section.

REFERENCES

[1] M. van Gerven and S. Bohte, *Artificial neural networks as models of neural information processing.* Frontiers Media SA, 2018.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, 2016, vol. 1.

[4] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[5] S. Pandiri, A. Al-Refai, and L. Lundberg, *Cloud Computing - Trends and Performance Issues: Major Cloud Providers, Challenges of Cloud Computing, Load balancing in Clouds.* LAP LAMBERT Academic Publishing, 2012.

[6] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security.* ACM, 2015, pp. 1310–1321.

[7] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[8] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[9] B. McMahan and D. Ramage, "Federated learning," https://research.googleblog.com/2017/04/federated-learning-collaborative.html.

[10] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. McMahan, T. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[11] V. Buterin, "Ethereum: A next generation smart contract & decentralized application platform," *Ethereum White Paper*, 2013.

[12] H. Diedrich, *Ethereum: Blockchains, Digital Assets, Smart Contracts, Decentralized Autonomous Organizations.* CreateSpace Independent Publishing Platform, September 2016.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," September 2009.

[14] V. Vielzeuf, S. Pateux, and F. Jurie, "Temporal multimodal fusion for video emotion classification in the wild," in *Proceedings of the 19th ACM International Conference on Multimodal Interaction.* ACM, 2017, pp. 569–576.

[15] S. E. Kahou, C. Pal, X. Bouthillier, P. Froumenty, Ç. Gülçehre, R. Memisevic, P. Vincent, A. Courville, Y. Bengio, R. C. Ferrari *et al.*, "Combining modality specific deep neural networks for emotion recognition in video," in *Proceedings of the 15th ACM on International conference on multimodal interaction.* ACM, 2013, pp. 543–550.

[16] G. Ye, D. Liu, I.-H. Jhuo, and S.-F. Chang, "Robust late fusion with rank minimization," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, 2012, pp. 3021–3028.

[17] N. Neverova, C. Wolf, G. Taylor, and F. Nebout, "Moddrop: adaptive multi-modal gesture recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 8, pp. 1692–1706, 2016.

[18] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[19] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[20] X. Yi, R. Paulet, and E. Bertino, *Homomorphic Encryption and Applications.* Springer, SpringerBriefs in Computer Science, November 2014.

[21] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme.* Stanford University Stanford, 2009, vol. 20, no. 09.

[22] Y. Masahiro, *Fully Homomorphic Encryption without bootstrapping.* LAP Lambert Academic Publishing, March 2015.

[23] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

TABLE VI

THE ACCURACY OF FADING CHANNEL DETECTION ACCURACY USING THE PRIVATE CNN-BASED LEARNING MODELS

Model	Detection Accuracy (%)		
	Without Encryption	Element-wise Encryption	Matrix-Pair-wise Encryption
Computing Contributor 1	93	91.6	91.7
Computing Contributor 2	93.1	92.4	92.0
Computing Contributor 3	93.2	92.3	92.1

TABLE VII

DETECTION ACCURACY ACHIEVED BY USING METALMODAL

Model	Detection Accuracy (%)	
	Element-wise Encryption	Matrix-Pair-wise Encryption
{1, 2}	94.5	94.5
{1, 2, 3}	95.5	95.0

TABLE VIII

THE IMPACT OF THE ELEMENT-WISE ENCRYPTION INTERFACE ON EXECUTION TIME

Model	Execution Time (ms)		
	Overhead for Encryption	Execution without Encryption	Execution with Encryption
Computing Contributor 1	7.7	60	126
Computing Contributor 2	7.6	57	123
Computing Contributor 3	7.6	59	125

[24] H. Zhou and G. Wornell, "Efficient homomorphic encryption on integer vectors and its applications," in *2014 Information Theory and Applications Workshop (ITA)*. IEEE, 2014, pp. 1–9.

[25] A. Yu, W. Lai, and J. Payor, "Efficient integer vector homomorphic encryption," May 2015.

[26] USA Patent provisional 62/663,287, 2018.

[27] Agorise, "c-ipfs: IPFS implementation in C. Why C? Think Bitshares' Stealth backups, OpenWrt routers (decentralize the internet/meshnet!), Android TV, decentralized Media, decentralized websites, decent." *Github.com*, October 2017.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[29] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[30] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.