VeriML: Enabling Integrity Assurances and Fair Payments for Machine Learning as a Service

Lingchen Zhao, Qian Wang, *Senior Member, IEEE*, Cong Wang, *Fellow, IEEE*, Qi Li, *Senior Member, IEEE*, Chao Shen, and Bo Feng

Abstract—Machine Learning as a Service (MLaaS) allows clients with limited resources to outsource their expensive ML tasks to powerful servers. Despite the huge benefits, current MLaaS solutions still lack strong assurances on: 1) service correctness (i.e., whether the MLaaS works as expected); 2) trustworthy accounting (i.e., whether the bill for the MLaaS resource consumption is correctly accounted); 3) fair payment (i.e., whether a client gets the entire MLaaS result before making the payment). Without these assurances, unfaithful service providers can return improperly-executed ML task results or partially-trained ML models while asking for over-claimed rewards. Moreover, it is hard to argue for wide adoption of MLaaS to both the client and the service provider, especially in the open market without a trusted third party. In this article, we present VeriML, a novel and efficient framework to bring integrity assurances and fair payments to MLaaS. With VeriML, clients can be assured that ML tasks are correctly executed on an untrusted server , and the resource consumption claimed by the service provider equals to the actual workload. We strategically use succinct non-interactive arguments of knowledge (SNARK) on randomly-selected iterations during the ML training phase for efficiency with tunable probabilistic assurance. We also develop multiple ML-specific optimizations to the arithmetic circuit required by SNARK. Our system implements six common algorithms: linear regression, logistic regression, neural network, support vector machine, K-means and decision tree. The experimental results have validated the practical performance of VeriML.

Index Terms—Verifiable computation, machine learning, secure outsourcing

1 Introduction

Machine learning (ML) has been widely used in a variety of fields such as disease diagnosis, risk prediction, and pattern recognition. Since ML tasks often tackle with massive data, especially in the training procedure, they require servers with strong computational capabilities. As a result, Machine Learning as a Service (MLaaS) has become a promising service paradigm that enables weak clients to train ML models or compute predictions in powerful cloud infrastructures.

Despite the well-understood benefits, there exist many serious concerns regarding MLaaS practices. “Pay as you use” is the common model of cloud computing billing. Users are charged according to the resources consumed by the outsourced task. However, as the service provider may have the motivation to overcharge clients for more profits, and its actual resource consumption is non-transparent to users, the billing is not always trustworthy. A typical example is that Amazon AWS has been reported to erroneously charge the user doubly or triply in the billing [1].

Moreover, the “uncertainties” of machine learning algorithms further deteriorate this problem. On the one hand, the workload of training a model cannot usually be determined in advance, because when the training converges is unpredictable. The convergence depends on the training data set, the learning parameter setting, convergence conditions, and random factors in the nature of model training. In this case, the server may claim that it has executed 10K iterations to train a model, but in fact it only executed 1K iterations, and the client cannot know the truth. On the other hand, the correctness of the returned result is unknown, and the user can only observe that the trained model or the prediction result seems “well-formed”. A famous case is that a startup Engineer.ai claimed they had built an AI-assisted app development platform, but it employed human engineers to complete tasks for cheating investors[2]. Another notable case is that machine learning-based automatic food-delivering robots in UC Berkleys campus were actually controlled by humans[3].

Therefore, verifying the correctness of the results as well as trustworthy accounting is vital and highly desirable. For wider adoption of MLaaS, we argue that the following three assurances are among the most important key desirables that MLaaS must satisfy.

- *Service correctness*: The client needs to be ensured that the ML tasks done by the service provider must work as

- *Lingchen Zhao and Qian Wang are with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan, Hubei 430072, China. E-mail: {lczhaocs, qianwang}@whu.edu.cn.*
- *Cong Wang is with the Department of Computer Science, City University of Hong Kong, Hong Kong. E-mail: congwang@cityu.edu.hk.*
- *Qi Li is with the Institute for Network Sciences and Cyberspace and Beijing National Research Centre for Information Science and Technology (BNRist), Tsinghua University, Beijing 100084, China. E-mail: qli01@tsinghua.edu.cn.*
- *Chao Shen is with the MOE Key Laboratory for Intelligent Networks and Network Security, Xi'an, Shaanxi 710049, China, and also with the School of Cyber Science and Engineering, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China. E-mail: chaoshen@mail.xjtu.edu.cn.*
- *Bo Feng is with the Khoury College of Computer Sciences, Northeastern University, Boston, MA 02115 USA. E-mail: feng.bo@northeastern.edu.*

Manuscript received 21 July 2020; revised 11 Mar. 2021; accepted 15 Mar. 2021. Date of publication 23 Mar. 2021; date of current version 3 May 2021.
(Corresponding author: Qian Wang.)
Recommended for acceptance by R. Prodan.
Digital Object Identifier no. 10.1109/TPDS.2021.3068195

intended, as if they were done in-house, and always produce correct ML predictions or correct trained ML models.

- *Trustworthy accounting*: In existing commercial MLaaS platforms, the bill is normally based on the consumed computing resources [4], [5]. Thus, the lack of full transparency at the service provider demands a strong assurance that the resource consumption claimed by the service provider indeed corresponds to the actual workload.

- *Fair payment*: The client should not obtain anything about the final MLaaS results (e.g., partial ML prediction result, sub-optimally trained ML models, etc.) prior to the payment. The fairness of exchange between the client's payment and the MLaaS result must be guaranteed.

Without these assurances, unfaithful service providers could return improperly-executed ML task results or partially-trained ML models while asking for over-claimed rewards. When designing schemes with strong assurance, we have to take into account all requirements together, so as to defeat sophisticated yet economically-incentivized attacks.

In the literature, *Verifiable Computation* (VC) is a powerful cryptographic building block which aims to verify the output of a deterministic function. The worker can produce a proof of his computation by representing the function as a circuit. An early attempt [6] invoked a common VC technique called the interactive proof protocol to verify the prediction results of neural networks. Yet, it did not address the cheating problem in MLaaS. First, it only verifies whether inferences are made. Second, it let the service provider reveal the model and results to the user, which violates our premise of fair exchange. Moreover, it can only verify linear models, and many widely-used non-linear operations such as divisions and comparisons are not supported.

Hence, our goal is to design a fair machine learning service system that can ensure the correctness of the results with high efficiency. To transform this high-level concern into a practical system, we need to address three main challenges:

1) Since existing VC techniques need to express the computation as a circuit, how to efficiently convert the machine learning algorithms to circuits is crucial for fast verification. We find that prior works in this regard are all unsuitable for VC, since their methods focus on the boolean circuit, which is expensive in calculating large numbers of multiplications [7], [8].
2) How to avoid the expensive cost when loading all the iterations and input variables in the circuit? An intuitive idea is to sample and verify a small part of the iterations. However, naive sampling allows a malicious server to only execute the sampled iterations and ignore the remaining ones. Besides, due to the sequential nature of the training, dependencies cross training iterations must be taken into account to ensure the correct check of the sampled iterations.
3) It is essential to ensure that neither of the two parties can get advantages in the payment is essential for a practical MLaaS. Directly invoking the VC technique still has risks of violating our fair exchange premise, where the training results from the server need to be revealed to the client for verification.

In this paper, we present VeriML, a practical outsourced machine learning framework, which can achieve integrity assurance and fair payment in training ML models, and detect misbehaviors with high probability. Different from prior arts on verifiable computation, the client of VeriML can be assured of not only the correct execution of ML tasks on the untrusted server, but also the fact that the claimed resource consumption by the service provider indeed corresponds to the actual workload. Unlike the prior method in [6], we build VeriML by leveraging the generic VC technique–succinct non-interactive arguments of knowledge (SNARK), for its lower computation and communication costs for the client, better expressiveness, and zero-knowledge property [9], [10]. Rather than directly applying SNARK to the entire ML task that often yields impractical costs, we focus on using SNARK only for randomly-selected individual iterative training procedures of ML for efficiency with tunable probabilistic assurance. To this end, we design a new commit-and-prove protocol, which avoids the expensive computation cost in proving large numbers of loop iterations by sampling. Its correctness is ensured by our commitment design that preserves complex dependencies between different iterations without information leakage.

As the case studies of VeriML, we focus on six typical machine learning algorithms: linear regression, logistic regression, neural network, SVM, K-Means and decision tree. We explore how to verify the execution of computing nodes in distributed learning. To reduce the computation cost in generating the proofs, we propose multiple optimizations to construct more efficient arithmetic circuits required by SNARK. To the best of our knowledge, VeriML is the first solution that achieves fairness and correctness in outsourced machine learning services. Our contributions can be summarized as follows:

- We present VeriML, the first outsourced machine learning framework that can exchange results of a paid service fairly and detect computation misbehaviors with high probability.
- To adapt machine learning algorithms to the VC technique, we design multiple circuit-friendly optimizations to verify expensive operations.
- We develop a new commit-and-prove protocol that can verify the loop iterations when training machine learning models with high efficiency. The detailed theoretic analysis demonstrates that our scheme can detect incorrect computations with high probability.
- We implement the VeriML framework with six popular ML algorithms: linear regression, logistic regression, neural network, support vector machine, K-means and decision tree on four real-world datasets to demonstrate its performance. Our results show that VeriML has a practical overhead as well as a negligible accuracy loss in training.

2 Related Work

Verifiable Computation. Verifiable computation is commonly used to verify the correctness of a function without re-executing it. Previous studies have been focused on three mainstreams: authenticated data structures (ADS), interactive proofs (IPs) and succinct non-interactive arguments of knowledge (SNARK). Among them, ADS has limited expressiveness and lacks the zero-knowledge property, which can only process certain computations such as polynomial evaluation [11] and graph queries [12]. IPs are implemented based on the sum-check protocol [13]. They can solve practical problems such as multiplication matrix [14] and SQL query [15] with high efficiency by avoiding expensive cryptographic operations. But it is complex to convert an IP protocol to zero knowledge [16], [17], [18] as needed by us. Compared to the above two solutions, SNARK transforms an arbitrary polynomial-sized function to a circuit to produce a short proof [18], [19], [20], and its succinct property makes it very suitable for weak clients. SNARK supports zero-knowledge proof, and has rich expressiveness, while the high cost at the prover's side can be alleviated by the powerful cloud.

Recently, verification frameworks without using the generic VC techniques have also been proposed. For example, [21] proposes to verify the results of SSE by smart contracts. TrueBit [22] presents a novel peer-review idea that introduces smart contract to judge the correctness of the results. However, achieving the judge contract requires the blockchain to store the trained models provided by multiple verifiers, which may cause expensive costs. Using trusted execution environment (TEE) such as Intel SGX is another orthogonal attempt, which might be complementary to our protocol [23], [24]. Using TEE demands additional trust of hardware vendors in the first place, which does not always hold in MLaaS. Besides, the limited size of the current enclave may cause expensive overhead in secure I/Os and encryptions/decryptions, especially when addressing large-scale ML training tasks. Nevertheless, TEE may be feasible for ensuring the integrity and fairness of MLaaS only if it can provide the zero-knowledge property like other arguments of knowledge protocols.

Privacy and Efficiency in Machine Learning. Prior works on privacy-preserving ML over encrypted data usually adopted homomorphic encryption (HE) and garbled circuits (GCs). These techniques are similar to VC as they also need to represent the computation task as a circuit. To our best knowledge, SecureML [8] was the first to train neural networks on encrypted data, and subsequent works following this direction have further obtained results with improved performance and/or accuracy [25], [26], [27]. Multi-party computation based methods can also help to ensure the integrity of training process, because the final results would be meaningless and also easily detected by the client if one of the parties does not execute the computation correctly. However, their execution costs are several dozen times higher than those of the original training tasks. Actually, in most of the existing commercial outsourced ML services, the clients choose to transfer their data to the server directly, and it implies that these data are not privacy-sensitive and do not need to be protected. Thus, it will unnecessarily incur a large computation burden to adopt MPC for integrity assurance.

Technically, the main difference between these works and ours lies in designing the circuit of machine learning algorithms, as we target at verifying the results. The most related work to ours is SafetyNets [6], which verified the correctness of the prediction of neural networks using IPs. However, SafetyNets assumes that both the server and the client are aware of the model and results during the whole process (including prediction and verification), and does not concern the training phase. Therefore, we cannot leverage SafetyNets to implement the training process with integrity assurances and/or realize the fair exchange.

Moreover, there exist some works which focus on improving the efficiency and accuracy of the ML algorithms. For example, Chen *et al.* proposed to combine data-parallel and task-parallel optimizations to improve the performance and scalability of the distributed random forest algorithm [28]. In [29], the authors presented a new clustering method that achieves low computational complexity and is suitable for large-scale data. In this paper, we only concern about some general optimizations for the classical ML algorithms, and these methods can be combined with our work to improve the performance in some special cases.

3 Preliminaries

The verifiable computation (VC) technique aims to enable a client C to verify the correctness of function F executed by server S with a given input x.

For an outsourced task, the client first runs KeyGen to generate an evaluation key EK_F and a verification key VK_F. It then sends EK_F to the server. The server executes Prove to produce the proof π and sends π and the result y to the client. The client then checks the proof to verify the correctness of y by executing Verify on the input x. The server and the client are referred to as the *prover* and the *verifier*, respectively.

Technically, the implementation of proof generation relies on SNARK. Its key point is to encode the user-defined computations as quadratic programs. The basic flow is to first compile the program from a high-level language to an arithmetic circuit,[1] and then use the circuit to construct a Quadratic Arithmetic Program (QAP), which includes three sets of polynomials $A := \{A_i(x)\}_{i=0}^m, B := \{B_i(x)\}_{i=0}^m, C := \{C_i(x)\}_{i=0}^m$ and a target polynomial $Z(x)$. Defining polynomial $P(x) = A(x)B(x) - C(x)$, and $Z(x)$ divides $P(x)$ iff $(c_1, \ldots, c_k)$ is a valid assignment for the circuit. The worker constructs $P(x)$ for the proof π, and the client can verify the correctness by checking if $Z(x)$ can divide $P(x)$. The *zero-knowledge* property can be easily drawn into SNARK with a negligible overhead by choosing three additional random samples $\delta_1, \delta_2, \delta_3$ and adding $\delta_1 Z(x), \delta_2 Z(x), \delta_3 Z(x)$ in the exponent to $A(x), B(x)$ and $C(x)$, respectively. The readers may refer to [30] for more details.

4 Problem Statement

4.1 Definition

In our system, a client C outsources a machine learning task to a server S with a training dataset D. S trains a prediction model M according to a certain ML algorithm and parameters. After the training phase, C submits challenges r to verify the execution of the learning algorithm, and in turn, S returns the corresponding proofs π without providing M. If all the proofs can pass the verification, C is convinced that S

[1] We do not consider boolean circuits because arithmetic circuits are more suitable for verifying the numerous multiplications.

has faithfully completed the training, and then pays for the ML service to obtain the model M. The core functions of our scheme are defined below.

Definition 1. *A fair machine learning service system allowing a client C to outsource the training algorithm F and a dataset D to the server S is a tuple of five algorithms:*

- $(EK_F, VK_F) \leftarrow \mathsf{KeyGen}(1^\lambda)$*:* is a probabilistic algorithm that takes as input a security parameter λ and outputs a public evaluation key EK_F and a verification key VK_F.
- $(r, \mathbf{I}) \leftarrow \mathsf{Compute}(D, F)$*:* is a deterministic algorithm that takes as input machine learning algorithm F and a dataset D and outputs the learning results r and a commitment $\mathbf{I}$.
- $(\pi, \mathbf{I}') \leftarrow \mathsf{Prove}(EK_F, F, x)$*:* is a deterministic algorithm that takes as input EK_F, an algorithm F, and data x, and outputs the corresponding proof π and auxiliary information $\mathbf{I}'$.
- $(0, 1) \leftarrow \mathsf{Verify}(VK_F, x, \mathbf{I}', \pi)$*:* is a deterministic algorithm that takes as input VK_F and outputs 1 if $F(x) = \mathbf{I}'$ and $\mathbf{I}' = \mathbf{I}$; 0, otherwise.

We say that VeriML is a secure protocol if the following properties are satisfied.

- *Completeness.* The probability that Verify outputs 1 (Accept) is 1 if S has completed the task.
- *Soundness.* The probability that Verify outputs 1 (Accept) is less than 2^{-l}, if S does not follow the protocol, where l is the bit length of the inputs.
- *Fairness.* C learns the witness iff he pays the fee, and S gets paid iff he has the correct result.

The key problem to guarantee fairness between S and C in this outsourced computation service is to efficiently verify the result correctness by a zero-knowledge proof. The definitions of correctness, security and efficiency of verifiable computations [30] are inherited.

4.2 Threat Model

We consider a generic setting for a cloud-based ML service, where a client C uploads a training dataset D to a server S, who runs the ML algorithm to train a model M. Like some prior works about verifying outsourced computation tasks, we assume that S is rational [31], [32], [33], [34], and its motivation is not only to deviate from the protocol, but also to earn additional economic benefits. Specifically, S may cut back on the training process to save computation and storage costs. Hence, we assume that if S has indeed executed the task correctly, it will not deliver a fake model after the verification. In Section 7, we will discuss how to relax this assumption by verifying the performance of the delivered model and designing appropriate pricing mechanisms. C is considered to be honest-but-curious, i.e., it may try to learn the trained model M before the payment. Obviously, *verifiability* is critical to the system, i.e., the client is allowed to verify the correctness of the trained model without knowing any information about it.

The main purpose of our work is to solve the problem in verifying the integrity of ML model training, i.e., proving that S has actually executed the specified computation task. Beyond that, we also aim to ensure the integrity of prediction services provided by S without revealing the ML model to C. In the literature, some works that study how to obtain information about the server's training data or by observing the prediction results, such as [35], [36], [37], [38], are outside the scope of this work.

5 Overview of VeriML

In this section, we first discuss the main design challenges, followed by an overview of our VeriML system.

5.1 Challenges

A naive solution to the problem of verifiable outsourced ML is to invoke existing VC protocols. C can construct a circuit that covers the whole learning process, and generate the corresponding key pairs. S evaluates the circuit and produces the proof. However, such a naive solution is infeasible in practice since ML algorithms usually require a large number of sequential loop iterations, and it is difficult to be represented by the circuit. Moreover, each iteration that consists of matrix multiplications and non-linear functions is extremely expensive for verification.

A straightforward modification to the above solution may be sampling several iterations for verification. A large enough sample size with uniformly-selected samples may help detect incorrect iterations with high probability. However, due to the nature of training, it is difficult to ensure that the proved iterations are the same as required. For instance, the client may request the proof for the 10K-th iteration, but the malicious server might still be able to cheat by executing an iteration with an arbitrary input. This means that whether the produced proof is actually corresponding to the 10K-th iteration needs be further proved.

Another limitation of existing VC methods is that they require the server to reveal the trained model to the client for proof verification. This allows the client to obtain the trained model without payment. Besides, if the client does not own the trained model, the server can arbitrarily forge proofs to pass the verification. Now we are facing a dilemma: *how can the correctness of the ML algorithm execution be verified without knowing the trained model?* And further, *how can fair payments be guaranteed?*

5.2 VeriML Outline

The core idea of VeriML is to make the training process retrievable, namely, the verifier can reconstruct the inputs and outputs for any specific iteration, and the retrieval process is verifiable. After the verification, a fair exchange of the payment and the trained model can be done via the blockchain. Fig. 1 depicts the architecture of VeriML. In summary, there are three phases: *Computation Phase*, *Verification Phase* and *Payment Phase*.

Computation Phase. S executes the machine learning task. To enable the retrievability of the training process, S needs to save additional auxiliary information, e.g., the intermediate states of the training process.

Verification Phase. After S completes the training, C sends multiple challenges to S for the proofs of specified iterations. In particular, C generates the key pairs and sends EK_F to S. Instead of verifying all iterations, C randomly

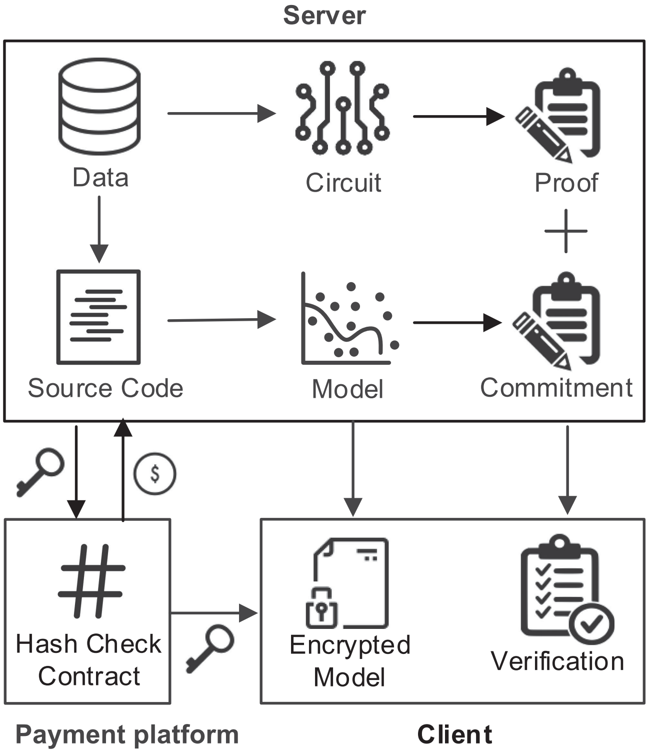


Fig. 1. Overview and workflow of VeriML.

samples a small subset of iterations as challenges, for which S produces the corresponding proofs. If all proofs can pass the verification algorithm, C considers that S has executed the ML task correctly with high probability.

To retrieve a specified iteration, S needs two inputs: the batch of the training data in the current iteration and the model's state in the previous iteration. The data batch selection can be agreed upon by the two parties. The model's state can be retrieved by re-executing the training process. To reduce the additional overhead, S saves some intermediate states of the model, referred to *checkpoints*. Then any state can be retrieved from the checkpoints by several iterations without cryptographic operations. Finally, S extracts a short identifier in each state during training and sends it to C for checking the correctness of the challenges.

Payment Phase. When all proofs pass the verification, C will pay S for the outsourced ML service. To ensure the fairness of exchange between C and S, a payment channel established by a trusted third party is needed. Recently, some works focus on using blockchain to replace the third party [21], [39], [40], [41], [42], and zero-knowledge contingent payment (ZKCP) [39] is a well-known solution that combines zero-knowledge proof and blockchain to ensure fairness in trading. Inspired by ZKCP [30], if there does not exist a trusted third party, we can also use blockchain to achieve this goal. The two parties first make an on-chain deposit. Then they use a smart contract to judge if S delivers the result. The contract will send the fees to S and return the deposit only if S presents the key of the encrypted delivery.

6 VeriML Design

In this section, we present our design of VeriML. We first show how to provide a fair machine learning service for linear regression, and then extend the protocol to support other ML algorithms which include logistic regression, neural network, support vector machines, K-Means and decision trees.

6.1 Linear Regression

We use a d-dimensional vector $\mathbf{w}$ to represent the parameters of the trained learning model. The value of $\mathbf{w}$ at the current iteration is referred to as a *state*. In each iteration, the SGD algorithm inputs a batch of samples $\mathbf{X}$, their labels $\mathbf{Y}$ and the current state $\mathbf{w}$. The state $\mathbf{w}$ is updated by penalizing the deviation between predicted labels and actual labels of the batch. The update equation is

$$\mathbf{w} = \mathbf{w} - \frac{\alpha}{b}\sum_{i=1}^{b}(\mathbf{x}_i\mathbf{w} - y_i)\mathbf{x}_i, \tag{1}$$

where b is the size of the batch, and α is the learning rate.

Decimal Arithmetic Operation. Existing implementations of VC by QAP [10], [18], [19], [43] do not support decimal numbers in arithmetic operations. To address this problem, we adopt the fixed point number representation. We restrict that each input has at most l bits of decimal points. Before invoking the circuit, we transform all inputs to integers by multiplying them by 2^l. However, the multiplication of two inputs will yield a product that has a different length. In linear regression, the right side of Eq. (1) will become $\mathbf{w} \times 2^l - \frac{\alpha}{b}\sum_{i=1}^{b}(\mathbf{X}\mathbf{w} \times 2^{2l} - \mathbf{Y} \times 2^l)\mathbf{X} \times 2^l$ by the fixed point number representation, in which the amplification of each term is inconsistent, and the operation will fail to produce the right result. Therefore, we need to apply appropriate scaling factors in the equation by multiplying it with a certain constant. After each iteration, the output needs to be truncated to l bits to serve as the input to the next iteration.

Circuit Construction. To verify the correctness in each iteration, we need to construct a circuit for prediction, and then calculate the mean square error (MSE) of the prediction for the current batch of data. In linear regression, the prediction only involves addition and multiplication. MSE is calculated as $y = \frac{1}{b}\sum_{i=1}^{b}(\mathbf{w}\mathbf{x}_i - y_i)^2$, which can also be implemented by addition and multiplication. If the batch size b is more than 1, the division $\frac{1}{b}$ can be transformed to one multiplication by a constant since the batch size is set in advance, e.g., if the batch size is 32, we can multiply the result by 2^{-5}.

Model Blinding. In existing VC schemes, C must know the input and output of each iteration, i.e., the intermediate state of the model, to perform verification. However, the state cannot be revealed to C since he may leverage it as a trained model and abort without payment. Therefore, we have to enable C to verify the correctness of each iteration without knowing the input and output states of the iteration. The input state can be preserved by the zero-knowledge property of SNARK, and we design a blinding scheme to mask the output state as an *identifier*. The blinding scheme should satisfy two requirements: 1) it is hard to retrieve the output state from the identifier, and 2) the collision probability of two different output states is small. The second requirement prevents S from producing a correct identifier with an incorrect input. Specifically, since the output state $\mathbf{w}'$ is only determined by the input state $\mathbf{w}$ and the batch $\mathbf{X}$, we need to ensure that it is hard for S to generate a fake state that can pass the verification with a given batch.

Intuitively, the hash function is a good choice for blinding the states. Nonetheless, implementing hash functions such as SHA256 in arithmetic circuit requires a lot of shifting and bitwise operations, which are much more expensive than arithmetic operations, and it significantly raises the computation cost of Prove. Moreover, the computation cost of hashes increases linearly with the dimension of the input state. To deal with this issue, we design a simple yet effective solution. We sum all elements in the output/input state vector $\mathbf{w}$. However, the straightforward summation has one shortcoming: the correctness of the summation result does not always ensure the correctness of the individual model parameters. To address this problem, we introduce an enhanced approach by using two private random coefficient vectors $\mathbf{v}$ and $\mathbf{r}$ generated by S to calculate the (random) weighted summation, and the vector $\mathbf{v}'$ used for protecting the output state $\mathbf{w}'$ is set as the multiplication of randomly shuffled $\mathbf{v}$ and $\mathbf{r}$. S would hash the inner product $\mathbf{w} \cdot \mathbf{v}$ and $\mathbf{w} + \mathbf{v}$ as the identifier, and make a commitment $H = \mathsf{SHA256}(r)$.

Remark 1. In the proposed commitment scheme, i.e., calculating the summation with a random coefficient vector, only computational hiding property provided by SHA256 is obtained, and binding between the model and the summation is missed since the random coefficient vector is predefined by a seed. However, the security of our scheme relies on the binding between the input and output of an iteration. It is ensured by the distinctness of the selected batches. We will give a more detailed security analysis of this design in Section 7.

Hyper-Parameter Encoding. The learning rate α has a significant impact on the training performance. Tuning α will affect the convergence speed of training, thereby affecting the number of iterations and the model's performance. In this paper, we mainly consider a fixed learning rate so that we can encode α into the circuit as a constant to reduce the proving cost. If α changes from iteration to iteration, to avoid recompiling the circuit and generating the key, we can treat α as an input to the circuit, which adds d multiplications for one iteration. The batch size b is another important hyper-parameter that can be encoded into the circuit. If the batch size increases by 1, $2d$ additional multiplications are required for each iteration, which greatly augments the proving time.

Iteration Verification. After constructing a circuit to verify the execution of each iteration, C can check all iterations to validate the integrity of the whole training process. However, the number of iterations may be huge, which leads to a prohibitively long time for proving correctness. To tackle this issue, we design a novel protocol that randomly samples a small subset of iterations to detect the misbehaviors during training.

Let N denote the total number of iterations that is claimed to be completed by S. In the training process, after finishing the ith iteration ($1 \leq i \leq N$), S saves the identifier $\mathbf{I}_i$. Upon completing the training process, S sends all identifiers $\mathbf{I}_1, \ldots, \mathbf{I}_N$ and $H_1, \ldots, H_N$ to C as a commitment for verification. S may adulterate identifiers to falsify his workload.

When C has received the commitment, he randomly samples a small subset $\{s_1, \ldots, s_m\}$ of m iterations ($1 \leq m \leq N$) as challenges. To produce the proof of the s_i-th iteration, S needs two inputs: the output state $\mathbf{w}_{s_{i-1}}$ of the previous iteration and the batch $\mathbf{X}_{s_i}$ of this iteration. Then S runs $\mathsf{Prove}(EK_F, \mathbf{X}_{s_i}, \mathbf{w}_{s_{i-1}}, \mathbf{r}_{s_i})$ to produce the proof and sends it to C. Finally, C checks if the produced proof π_i can pass $\mathsf{Verify}(VK_F, \mathbf{I}_{s_{i-1}}, \mathbf{I}_{s_i}, \pi_{s_i}, H)$ and determines if $\mathbf{I}_{s_i}$ is the same as the corresponding identifier in the commitment. The circuit directly outputs $\mathbf{w} + \mathbf{v}$, $\mathbf{w}' + \mathbf{v}'$, $\mathbf{w} \cdot \mathbf{v}$, $\mathbf{w}' \cdot \mathbf{v}'$, and C checks if their hash is the same as the identifier. Note that the circuit can output the preimage of the identifiers directly because all the zero-knowledge values are protected by the random numbers. Besides, the proving phase can also be accelerated in parallel by utilizing the cloud server's multiple CPUs or cores, because generating proofs for different iterations are processed independently.

Saving all intermediate states inevitably induces a high storage cost for S. Therefore, we propose to set checkpoints to retrieve the states. Specifically, we partition the N iterations into k groups, and thus each group contains N/k states. S only saves the output state of the first iteration in each group as a checkpoint such that the ith state in a group can be retrieved by re-conducting i iterations from the checkpoint. Note that S can tune the parameter k to make a tradeoff between the storage and retrieving costs. The retrieving process does not involve any cryptographic operations, i.e., generating keys or producing proofs, and thus the additional computation cost is acceptable.

Obviously, the core of our scheme is to reproduce the identifiers by VC. To achieve this, in the training and verification phases, the chosen data batches need to be consistent. In general, the inputs are chosen by randomly shuffling the whole dataset, so we can let C assign a random seed and send it to S. For each epoch, as long as the two parties use the same seed and algorithm to shuffle the dataset, they can reproduce and verify the identifiers later. Note that C has to check the distinctness of the selected batches. Otherwise, S will have advantages to forge identifiers.

Payment. The payment process has to ensure a fair exchange, i.e., to prevent either S or C from cheating one another. More specifically, a malicious C would like to obtain the trained model without paying anything, while a malicious S expects to get the payment without executing the whole training process.

A trusted third party is proved to be indispensable in achieving fair exchange [44]. Hence, if there exists a trusted third party, e.g., a credit card company or a trading platform, it can help to complete the exchange. When the trusted third party is unavailable, additional designs are needed. The Zero-Knowledge Contingent Payment (ZKCP) protocol, which is designed based on the blockchain, is a possible solution. In the protocol, S encrypts the file f with the symmetric key k and sends both $\mathsf{Enc}_k(f)$ and $h = \mathsf{SHA256}(k)$ with a proof π to C. If π can pass the verification algorithm run by C, S can prove that k is actually the key of $mathsfEnc_k(f)$ and the preimage of h. Then C posts a transaction on the blockchain to pay to anyone who can provide a preimage of h to obtain k, and S can obtain the money by presenting the preimage.

Inspired by ZKCP, we can build a trivial protocol to achieve the fair exchange. First, S encrypts the final state of

VeriML Protocol for the Training Service

- **Setup**:
    1) The client C sends the parameters of learning algorithm to S, which includes the learning rate α, the batch size b, and the threshold of convergence t. After the agreement, C compiles the verification circuit F to the QAP program locally.
    2) With the security number λ, C generates the key pairs $(EK_F, VK_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$ and a random seed s to be used for choosing batch samples. Then C sends the seed s to S.
- **Computation Phase**:
    1) For each iteration i, S selects the batch samples by seed s.
    2) Taking state $\mathbf{w}_{i-1}$ of the previous iteration and the data batch as input, S outputs an updated state $\mathbf{w}_i$ and saves its identifier $\mathbf{I}_i$.
    3) After every N/k iterations, S saves the state as a checkpoint.
    4) If the difference of accuracy between epoch e_m and e_{m-1} is less than the threshold t, S terminates the training process and sends identifiers $\mathbf{I}_1, \ldots, \mathbf{I}_m$ to C.
- **Verification Phase**:
    1) C sends the circuit and EK_F to S.
    2) S checks the correctness of the circuit.
    3) C randomly chooses m iterations $s_1, \ldots, s_m$ for verification and sends the set of indexes to S to challenge the corresponding proofs $\pi_{s_1}, \ldots, \pi_{s_m}$.
    4) S locates the checkpoints and retrieves states $\mathbf{w}_{s_1-1}, \ldots, \mathbf{w}_{s_m-1}$. Then S produces proof $(\mathbf{I}'_{s_{i-1}}, \mathbf{I}'_{s_i}, \pi_{s_i}, H) \leftarrow \mathsf{Prove}(EK_F, F, \mathbf{w}_{s_{i-1}}, \mathbf{X}_{s_i}, \mathbf{r}_{s_i})$ for each i, and sends the outputs to C.
    5) C selects the inputs of samples also by the random seed s, and then runs $v_{s_i} \leftarrow \mathsf{Verify}(VK_F, \mathbf{X}_{s_i}, \mathbf{I}'_{s_{i-1}}, \mathbf{I}'_{s_i}, \pi_{s_i}, H)$ for all selected iterations. If there exists $v_{s_i} = 0$, C outputs Reject; otherwise, C compares $\{\mathbf{I}'_{s_{i-1}}, \mathbf{I}'_{s_i}\}$ with $\{\mathbf{I}_{s_{i-1}}, \mathbf{I}_{s_i}\}$ in the commitment. If $\forall\ 1 \leq i \leq m$, the commitments and the verification results are the same, C outputs Accept, otherwise Reject.
- **Payment Phase**: If C outputs Accept after the verification phase, the two parties enter the payment phase. (If there exists a trusted party, this phase can be omitted.)
    1) S encrypts the trained model $\mathbf{w}$ by a symmetric key k, and then sends $\mathsf{Enc}_k(\mathbf{w})$ and $h = \mathsf{SHA256}(k)$ to C.
    2) C posts a transaction T on the blockchain to pay the pre-determined fee f to the party who presents x and x' such that $\mathsf{SHA256}(x) = h$ and $\mathsf{SHA256}(x') = H$.
    3) S presents string z and z' to T. If $\mathsf{SHA256}(z) = h$ and $\mathsf{SHA256}(z') = H$, T sends the fee to S. Otherwise , T returns the payment to C.

Fig. 2. The VeriML protocol for the training service.

the trained model using a symmetric key k and then sends $\mathsf{Enc}_k(\mathbf{w})$, $H = \mathsf{SHA256}(r)$ and $h = \mathsf{SHA256}(k)$ to C. C posts a transaction on the blockchain to transfer the payment to anyone who reveals a preimage of h and H. In this way, C cannot decrypt the trained model without payment via the transaction on the blockchain, and S cannot obtain the payment without passing the verification or providing the key to C.

Prediction Service. We also consider the case where S provides the prediction service to C instead of training a model. To verify the prediction results, we can construct a similar circuit, but do not have to yield multiple identifiers to help check the correctness of each iteration. Here, S should not directly reveal the prediction result to C for verification, since C may get away with the result without paying anything. Hence, we transform the verification process to a zero-knowledge proof that sets the prediction results as a witness, and compares whether the witness equals the result of forward propagation calculated by the circuit. If it does, C can confirm that S actually generates the correct prediction results.

We present the protocol for the training service by summarizing the above constructions in Fig. 2. An adapted protocol for the prediction service is given in Fig. 3.

6.2 Logistic Regression

Compared with linear regression, logistic regression faces the main challenge of computing the sigmoid activation function $f(x) = \frac{1}{1+e^{-x}}$, since the division and exponentiation are not supported by an arithmetic circuit. Prior works have presented various approaches to approximate the sigmoid function. In this section, we will discuss the differences of these approaches in terms of efficiency and accuracy, and propose a more feasible solution for the QAP application.

Piecewise and polynomial approximation are two mainstream approaches to implementing sigmoid functions in prior works [8], [45], [46]. The piecewise method, which relies on comparisons, is much more expensive than arithmetic operations in QAP, since one comparison between two l-bit integers requires a *split* operation that consumes

VeriML Protocol for Prediction Service

1) C generates the key pair $(EK_F, VK_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$, and then C sends EK_F and data $\mathbf{X}$ to S.
2) S runs the prediction algorithm to compute the result. Then S produces the proof $\pi \leftarrow \mathsf{Prove}(EK_F, F, w, \mathbf{X})$ taht takes the result as a witness, compares if the witness is equivalent to the output of the circuit, and then sends the produced proof π to C.
3) C runs $v \leftarrow \mathsf{Verify}(VK_F, \mathbf{X}, \pi)$ and outputs Reject if the output is 0. Otherwise, C sends the request for the result. (If there exists a trusted party, the following steps can be omitted.)
4) S encrypts the result m as c by a symmetric key k: $c = \mathsf{Enc}_k(m)$, and sends c and the hash value $h = \mathsf{SHA256}(k)$ to C.
5) C posts a transaction T on the blockchain (or a trusted party) to pay the fee f to the party who presents x such that $\mathsf{SHA256}(x) = h$.
6) S presents the string z to T. If $\mathsf{SHA256}(z) = h$, T sends the fee to S, otherwise gives the refund to C.

Fig. 3. VeriML protocol for the prediction service.

$l + 2$ constraints, while one multiplication between two integers only consumes one constraint. Therefore, the polynomial approximation is obviously more favorable.

Taylor expansion is a classical method to approximate nonlinear functions, and its accuracy highly depends on the degree of the polynomial terms. The higher the degree of the polynomial term, the better the approximation performance. But the computation cost will be higher, and it is easy to exceed the finite field. Inspired by [46], we use the *Remez* algorithm to implement the approximation with high efficiency and accuracy. However, the Remez method is only suitable for a certain range of the input because of the unbounded tails, and the input beyond the proper range may affect the accuracy of the approximation, hence we need to set an appropriate range to calculate the approximated polynomial.

Setting the degree of polynomials as three and the approximated range of x as [-5, 5], the approximation of Remez is $f(x) = -0.004x^3 + 0.197x + 0.5$, and that of the Taylor expansion is $f(x) = -\frac{1}{48}x^3 + 0.25x + 0.5$. It can be seen that the Remez method is closer to the original sigmoid function in a wider range. Experimental results about the training accuracy of the approximations are presented in Section 8.

6.3 Support Vector Machine

Besides the SGD-based methods, support vector machine (SVM) is another classic and popular machine learning algorithm to solve classification problems. The training process of SVM can also be represented as sequential loop iterations, so we can transfer it into the VeriML framework naturally. In this paper, we consider the most representative case of binary classification proposed in [47]. The details of SVM are omitted due to the space limitation.

The main construction of training a SVM model is the same as the prior methods, i.e., using a small batch of data to update the model after each iteration until the objective function converges. Specifically, apart from the basic additions and multiplications, each iteration includes two divisions, one euclidean projection and $b + 1$ comparisons. As discussed before, verifying divisions and square roots can be transformed to the multiplication operations using the pre-computed results. Therefore, we can construct the arithmetic circuit straightforwardly.

6.4 K-Means

VeriML can be scaled with clustering, a type of task which expects to partition multiple data samples into several clusters, because the process of training a clustering model also consists of sequential loop iterations. Here we use the most typical clustering algorithm–K-Means to demonstrate the protocol design.

In the beginning, C randomly chooses k centroids to represent the clusters. In each iteration, the K-Means algorithm assigns each training sample to the cluster closest to it and then uses the average of all the samples in this cluster to update the centroid. Here we choose to use a small batch of data in each iteration to reduce the circuit size. According to the prior results [48], this operation will not cause a significant impact on the accuracy. Since the centroids are represented by vectors, the commitment can be generated by the random coefficient vector as well.

Verifying the Closest Distance. The main cost of executing the circuit lies in finding the closest distance of the k centroids. When the batch size is b, each iteration needs to execute bk comparisons, which involve large computation overhead. Here we avoid the comparisons by checking the correctness of the candidate closest distance given by the server. Specifically, for each data sample, the circuit takes the candidate closest distance as an input. Then the circuit executes the subtractions between the candidate closest distance and all the k previously-obtained distances, respectively. If the circuit finally outputs only one 0, and all the other results are negative, the candidate closest distance is considered as a correct one. As the maximum value is unknown to the client, the outputs will not reveal the private values.

6.5 Decision Tree

Decision tree and its variants are significantly different from other machine learning methods discussed above. This is because training or using a tree is composed of comparisons, rather than additions and multiplications. Intuitively, verifying the correctness of the structure of a trained tree is equivalent to checking two conditions: 1) for each internal node, whether the partition is done based on the largest information gain, and 2) whether the data samples belonging to this node are actually composed by its children nodes. The sampling strategy can still be adopted to verify the integrity of the whole tree, and the comparisons can be verified efficiently by the technique we proposed in verifying K-Means. Since the number of data samples directly affects the correctness of finding partitions, the batch strategy is no longer applicable. Now we are facing two challenges: 1) how to reduce the circuit's I/O caused by traversing all the training data, and 2) how to avoid revealing the tree to C in the verification phase.

Compressing the Inputs. Instead of using batches to split the data horizontally, we use histograms to reduce the input size vertically. LightGBM [49] proposed representing inputs with histograms to accelerate the training of decision trees. Inspired by this idea, in our design, C buckets feature values into multiple bins before the training process. Concretely, for each feature, C first converts its field to bins, traverses all the data to construct the histogram, and sends it to S. S can only use the histogram to calculate information gains and find partitions. Assume that there are n data samples. Each sample has d features, each of which is bucketed into k discrete bins. Using histogram can thus reduce the number of inputs from nd to kd. In the meanwhile, verifying if a node is correctly partitioned can be transformed to verifying if its histogram is equivalent to the addition of its child nodes' histograms.

Commitments for Decision Tree. Compared with other ML algorithms, training decision tree does not include the iterative optimization process. This makes the previous method that blinds the states of the model no longer applicable. Instead, we observe that the histogram is updated after each partition, and the updates cannot change the structure other than the values of bins, so we can blind the histogram with the same commitment method described before and check if the histograms of sampled nodes are the same as the commitment. Such a commitment may reveal the structure of the tree to C, and this can be fixed by adding dummy nodes to fill the tree to be "perfect", i.e., all the leaves have the same depth, and all the internal nodes have the same degree.

Computing predictions by decision tree can only be implemented by multiple comparisons. Note that for categorical features, comparing the features with partitions can be transformed to checking the equalities, which has lower computation costs.

6.6 Neural Networks

To efficiently apply VeriML to neural networks, we design an *inversed verification* method to reduce the size of the circuit by utilizing pre-computed results.

Matrix Multiplication. Traditional matrix multiplications using circuit has a complexity of $O(n^3)$, which is very time-consuming. Observations in prior works show that verifying the correctness of results is much cheaper than computing the results forwardly [9], [19], [20]. For example, it is hard to implement the function of verifying $c = a/b$, but instead we can verify $a = b \times c$ efficiently. Following this rationale, we use Freivald's algorithm [50] to inverse the forward computation when constructing the circuit. Freivald's algorithm is a probabilistic randomized algorithm for verifying matrix multiplications. Assume that we have three $n \times n$ matrices A, B and C. By using a uniformly-sampled $n \times 1$ random vector r over field $\mathbb{Z}_s^n$, the correctness of $AB = C$ can be reviewed by verifying whether $A(Br) = Cr$ stands. The false positive rate is $1/(s+1)$. If we selecting a large field, the probability of a false positive can be reduced to a negligible value. Since the random r is selected after generating the commitment, it can be selected in advance and encoded in the circuit as constants to save the computation cost.

With the help of Freivald's algorithm, the verification requires only $O(n^2)$ multiplications. For neural networks, let n_i denote the number of neurons in the ith layer. The number of matrix multiplications in one layer descends from bn_in_{i-1} to bn_{i-1} (with $bn_{i-1} + n_in_{i-1}$ multiplications by constants). For a neural network with β matrix multiplications, as these multiplications are independent, the false positive rate for one iteration is less than $1/(s+1)^\beta$, which is negligible.

Softmax Function Verification. The softmax function $f(x_i) = \frac{e^{-x_i}}{\sum_{i=1}^{\kappa} e^{-x_i}}$ is used for multi-class classification, where κ is the number of classes. To verify the function, we have to tackle the difficulties of computing exponentiation and division in the circuit. The output of softmax is a probability distribution so that all results are non-negative. Therefore, we can adopt the square function to replace exponentiation operations. Unlike other activation functions that can be approximated into division-free forms, the division operation in the softmax function is inevitable. Since that division is not supported by SNARK (division by constant can be transformed to multiplication), we adopt the strategy of inverting verification that checks the equality of the dividend and the product of the divisor and the result. For each data sample, S calculates the results of softmax and feeds them as witness to the circuit. For a batch of data, the input to the circuit is a $b \times \kappa$ matrix, and the circuit needs to perform bm additional multiplications.

Discussions. In a typical SNARK implementation, the arithmetic circuit operates over a 254-bit field. The continuous multiplications of multiple layers make the length of the results increase rapidly, which may lead to an overflow. Although the verification inversion method can mitigate the overflow due to fewer multiplications, truncating the multiplication results before feeding them to the circuit is still needed. To avoid the problem of inconsistent bit lengths of the input and the last iteration's output, we conduct an additional check to see if the difference between the two values is less than 2^{-l}. In addition, some other tricks that utilize randomness in training neural networks, such as dropout, are easy to be implemented by using a consistent random seed.

Generally speaking, ML algorithms consist of basic computation operations including addition, subtraction, multiplication, division, comparison, square, and other nonlinear operations (e.g., root, logarithm, etc.). Therefore, the support of these basic operations means that VeriML is applicable to other models with more complex structures and calculations. Specifically, arithmetic operations (e.g., addition, subtraction, multiplication, and square) can be verified directly by the circuit, divisions can be represented by multiplications, comparisons can be converted into observing the sign bits of the outputs, and non-linear operations can be approximated by polynomials. In this paper, we choose six well-known and representative algorithms, which cover the typical ML tasks (regression, classification, and clustering). Most of the other algorithms (e.g., random forest, GBDT, KNN, etc.) can be regarded as the variations of them and verified by the proposed building blocks.

6.7 Distributed Learning

Considering the communication cost of transmitting a large amount of data or the privacy risks in revealing data to other entities, the training data can be stored in multiple nodes. In this scenario, a distributed learning method is needed to make a full use of all the data.

A common paradigm of distributed learning is as follows. Every computing node runs the training algorithm locally on its data and uploads the model parameters to an aggregator, which aggregates the uploaded results to obtain a global model. Then these nodes download and update the global model. This process will repeat many times until the training converges. There exists a similar problem that a lazy computing node may not contribute to the training process and only upload fake results.

With some slight changes, the proposed scheme can also support the verification of the integrity of distributed learning. As the iterative optimization process is running on every node, the aggregator plays the role of the verifier, and the nodes acts as the prover. Similarly, the node needs to make commitments for every iteration's output and sends them to the aggregator. The aggregator randomly samples some iterations and asks for the proofs. Then the node uses the corresponding training data and model parameters to generate proofs. The aggregator will check if all the proofs can pass the verification algorithm, and the inputs and outputs are corresponding to the prior commitments. This method can also help to detect misbehaviors of uploading tampered or incorrect models [51], [52].

The main difference between verifying outsourced learning and distributed learning is that the model parameters are not private to the aggregator, because they will be used to generate the global model. Therefore, the nodes also need to send the parameters of the sampled iterations to the aggregator, and these parameters should no longer be zero-knowledge inputs. By contrast, the verifier may not know the training data collected by the nodes themselves. In this case, the data should be set as zero-knowledge inputs.

As setting the input as zero-knowledge will not significantly affect the computation cost (fixed 213μs to KeyGen and 0.1 percent to Prove [10]), we omit the additional experimental results about verifying the specific ML algorithms.

7 Security Analysis

7.1 Security Proof

Theorem 2. *Assuming there exists no more efficient algorithm that can output the correct result than the training algorithm, the proposed protocol VeriML is secure if the properties of completeness, soundness, and fairness are satisfied simultaneously.*

Proof. First, we demonstrate the rationality of this assumption. Intuitively, if there exists a more efficient machine learning algorithm, it should be known by anyone, including C. Hence, C will ask S to use this algorithm, generate the corresponding circuits and keys, and admit the corresponding bills. The above process can prevent S from lying about the used algorithm. Even if there exists an algorithm that is more efficient only on some special inputs/outputs, it will not break this assumption because the input of every iteration is assigned by C, i.e., S cannot take this advantage to obtain the results.

Completeness. The completeness of the setup, computation and verification phases depends on the completeness of the underlying verifiable computation scheme. The completeness of the payment phase depends on the correctness of SHA256 and the third party (the consensus mechanism of blockchain).

If both S and C faithfully follow the protocol, S can pass the Verify algorithm and earn the service fee by presenting the preimage of the key to C for decrypting the trained model, and C can obtain the trained model or prediction service by posting the payment transaction.

Soundness. A forged training workload will be accepted by C iff all the sampled iterations can pass the verification algorithm. Because the outputs of VC can always be considered as the ground truth with the given inputs, a cheating server S aims to provide a pair of models $\mathbf{w}_{i-1}$ and $\mathbf{w}_i$ such that the outputs of VC are the same with the commitment. If the probability for achieving this is less than 2^{-l}, the soundness property can be proved. We show that if S can pass Verify without the correct execution, there are only two ways for S to cheat.

As the randomly generated coefficient vector $\mathbf{v}$ is predefined before making the commitment, S can always find either $\mathbf{w}^*_{i-1}$ or $\mathbf{w}^*_i$ that corresponds to the committed identifier. Hence, our aim is to prove that it cannot forge the other one without executing the computation correctly.

Case 1. The cheating S directly forges the identifiers $\mathbf{I}_{i-1}$ and $\mathbf{I}_i$ without using the models when generating the commitment. Since the coefficient vector is uniformly random, the distribution of identifiers is uniformly random. Therefore, S can only forge identifiers by generating random strings.

When the ith iteration is sampled, S can find an arbitrary vector that has the committed identifier as the input. Obviously, the forged identifier of the output will be detected since it is not obtained from the current input. If the bit lengths of the output parameters $\mathbf{w}$ and the random coefficients $\mathbf{v}$ are both l, their dot product extends the preimage of the hash to $2l$ bits, and the probability that the forged identifier is the same as the ground truth computed by VC according to the current input is 2^{-2l}.

Case 2. The cheating S forges the identifiers by finding possible values of $\mathbf{w}^*_{i-1}$ or $\mathbf{w}^*_i$. We assume that the cheating S can find a way to estimate the input and output states without executing the training algorithm when generating the commitment. We do not make any assumption about its approach, e.g., it may utilize some background knowledge about the model such as the distribution of the parameters. Yet, according to the basic assumption that there exists no way to obtain the result without the correct computation, the forged model and the ground truth of the output computed by VC according to the current input should be different.

To ensure that $\mathbf{w}^*$ and the ground truth $\mathbf{w}$ have the same identifier, the server needs to find a $\mathbf{w}^*_{i-1}$ and $\mathbf{v}^*_{i-1}$ that can make $\mathbf{w}^*_{i-1} + \mathbf{v}^*_{i-1}$ and $\mathbf{w}^*_i + \mathbf{v}^*_i$ corresponds to the commitment. $\mathbf{w}_i$ can be represented by d equations about $\mathbf{w}_{i-1}$ and $\mathbf{X}$, and the two identifiers include $2d$ equations about $\mathbf{w}_{i-1} + \mathbf{v}_{i-1}$ and $\mathbf{w}_i + \mathbf{v}_i$. Hence, S needs to find a solution about $2d$ unknowns. Then, the solution has to guarantee

$$\mathbf{v}_{i1}(\mathbf{w}_{i1} - \mathbf{w}^*_{i1}) + \mathbf{v}_{i2}(\mathbf{w}_{i2} - \mathbf{w}^*_{i2}) + \cdots + \mathbf{v}_{id}(\mathbf{w}_{id} - \mathbf{w}^*_{id}) = 0. \tag{2}$$

Since $\mathbf{w}^*$ does not equal to $\mathbf{w}$, there exists j such that

$$\mathbf{v}_{ij} = \frac{\mathbf{x}(\mathbf{w}_{ij} - \mathbf{w}^*_{ij}) - \mathbf{v}_{ij}(\mathbf{w}_{ij} - \mathbf{w}^*_{ij})}{\mathbf{w}^*_{ij} - \mathbf{w}_{ij}}. \tag{3}$$

Since the identifier is uniformly random, the value of the right hand side in Eq. (3) is uniformly random. If the bit length of $\mathbf{w}$ is l, the probability that the solution satisfies Eq. (3) is 2^{-l}.

There may exist some special cases that will discard some steps in an iteration. For example, in the SGD algorithm, if S can find a model $\mathbf{w}^*$ whose outputs are exactly the same as the labels of the data batch, i.e., $\mathbf{w}^* \times \mathbf{x} = \mathbf{y}$, $\mathbf{w}^*$ will not be updated in this iteration. In other words, S can directly output $\mathbf{w}^*$ without any computations. Such an attack is easy to be detected by VC, because S knows the existence of the detection. Thus, the soundness property of our system is satisfied.

If the claimed total number of iterations is N, the proportion of genuine identifiers is t (i.e., the proportion is $1-t$ for corrupted ones), and C randomly samples c different iterations, the probability that all the sampled iterations are genuine is

$$p = \frac{\binom{tN}{c}}{\binom{N}{c}} = \frac{(tN+1-c)(tN+2-c)\cdots(tN)}{(N+1-c)(N+2-c)\cdots N}, \tag{4}$$

which means that the upper bound of this probability is p.

When a fake identifier passes the verification, this upper bound will be relaxed. Given the definition of soundness, if all the data and parameters have l bits, the probability that a fake identifier is exactly the same as the genuine one is less than 2^{-l}. Hence, the upper bound is relaxed to

$$\epsilon = \sum_{i=0}^{c} \frac{\binom{tN}{c-i}}{\binom{N}{c}} \frac{1}{2^{li}} < p + (1-p)\sum_{i=1}^{c} \frac{1}{2^{li}} < p + \frac{1-p}{2^l - 1}. \tag{5}$$

For instance, if S claims 100K iterations and performs only 70K iterations ($t = 70\%$), C only needs to verify 9 or 13 iterations to detect the misbehavior with a probability higher than 95 or 99 percent, respectively.

Fairness. A malicious S has incentives to forge proofs or identifiers in the commitments. Since the program and the evaluation keys are provided by C, it is easy to detect fake proofs. The probability that a fake commitment passes the verification is negligible after sampling multiple iterations. Therefore, S is prevented from forging the training process. Once it is detected, it will not get paid. Also, S will faithfully present the preimage of the key k to receive the payment. As a result, the protocol ensures that S will deliver the trained model and claim the real training workload.

A malicious C is motivated to learn about the trained model without payment, and C can only obtain the output of VC because of the zero-knowledge property of the underlying VC protocol. It is also difficult for C to manipulate EK_F to infer the exact value of the witness (we will demonstrate the reasons shortly after.) If c expects to infer $\mathbf{w}_{i-1}$ or $\mathbf{w}_i$, he needs to find a solution that satisfies the two commitments. Moreover, as the commitments include $2d+2$ equations, and $\mathbf{w}_{i-1}$ and $\mathbf{v}_{i-1}$ consist of $2d$ unknowns, if there are more than 3 different elements in r, C cannot obtain the unique solution of $\mathbf{w}_{i-1}$. Other malicious behaviors, such as claiming that a correct proof is incorrect or posting an invalid transaction on the blockchain, cannot help C to learn the model. □

7.2 Discussions

Discussions About the VC Component. In [40], the authors proposed that C may modify the common reference string (CRS), i.e., the EK_F and VK_F, to learn information from proofs. A malicious C can check whether a value in the witness is the exact one or not. For instance, in the pay-to-sudoku service, the client can find out the exact value for a Sudoku cell with a probability of $1/9$.

This attack is derived from the requirement of the trusted setup in zkSNARK. In the outsourced computation scenario, it can be easily fixed by generating the CRS collaboratively, e.g., using the coin-tossing protocol to choose the random numbers needed in generating the keys.[2] If we only focus on such an attack, it is actually unavailable in machine learning services because of the difficulty in finding the witness represented by dozens of bits. Specifically, if a parameter in the witness has l bits, the probability of finding its exact value is $1/2^l$, which is negligible. Therefore, C is allowed to choose the CRS in our scheme. More importantly, we emphasize that as a case study, the underlying used zkSNARK component in our design can be replaced by any other similar zero-knowledge argument of knowledge systems, e.g., Hyrax [53] and Bulletproof [54], which may have better performance in efficiency or security.

Several other attacks have been proposed to break the original ZKCP protocol [40]. If the purchase is a service (not goods), e.g., an audit of online file storage, C can infer from the proof that the service is correct, and then abort the protocol without payment. However, for the outsourced ML service, the client aims to obtain the trained model or prediction results rather than a simple answer of yes or no. The proof only allows C to certify the correctness of the service but will not reveal any additional information. Hence, there is no incentive for C to abort the protocol after verifying the proofs.

Finally, C needs to ensure that the results delivered by S are actually encrypted by the key. This may require symmetric encryption schemes such as AES [55], and the circuit implementation of them will lead to a high cost (4.2×10^6 constraints only for 300 blocks [20]). For instance, the simple three-layer fully-connected neural network with 128 hidden neurons consumes 25,408 blocks (if each parameter has 32 bits), which is too expensive. Recently, FairSwap [42] proposed to utilize the Merkle tree to verify the hash value of a large file without using the zero-knowledge proof.

[2] Note that the protocol about sampling iterations is not under sequential composition as proving every iteration is independent. Therefore reusing a single CRS multiple times is secure.

Combining this technique with VeriML may help achieve stronger security guarantee.

In summary, the attacks that can break the ZKCP protocol will not affect our scheme, because it is difficult to guess the continuous values in machine learning. Amd what we focus on is the verification of the computation workload of the training process, rather than verifying the existence of just a service or a file. Moreover, since the underlying VC protocol is independent with our system, these attacks can be better defended by other building blocks that do not need the trusted setup, such as [17], [18].

Discussions About the DoS Attack. Apart from the fairness concern of ensuring a fair exchange of service and payment between the client and the server, another issue is that parties may prematurely abort from the protocol. For instance, a malicious server may be interested in obtaining the data from the client but abort before or during the computation phase, while a malicious client may launch a DoS attack by aborting the protocol before the payment phase or lying about the verification to consume the computing resources of the server.

One possible solution is to ask the client and the server to make certain deposits in advance. If the protocol is followed through, each party will be refunded their deposits. Otherwise, if one party prematurely aborts the protocol, this party will lose the deposit. Intuitively, the deposit of the malicious party should be used to compensate for the other party. However, it is difficult for the smart contract to detect certain malicious behaviors, e.g., if the client outputs Reject in the verification phase, it is hard to tell whether the server fails in the verification or the client lies about the verification. This is partly because the cost of implementing complex operations on a smart contract is very expensive. In [42], the authors proposed to mitigate the risk from a malicious client by making the server do some pre-computation activities. But in MLaaS, the server cannot perform any pre-computation before receiving the data from the client. Therefore, the traditional deposit and refund strategy cannot well solve the malicious abort problem in MLaaS, and this can be an interesting open problem.

The Correctness of the Training Algorithm. Another potential malicious behavior is increasing the task deliberately, but still executing the computations, e.g., choosing suboptimal algorithms to slow down the convergence speed, or keeping training after meeting the pre-defined termination condition to charge more. As the condition of maintaining integrity is paying more costs, preventing this behavior by setting the pricing mechanism is an appropriate solution. For example, if the server can complete the task in half of the time, the rewards are higher than the half of the prior one. This strategy can increase the rate of profit, and incentivize the server to complete the task as soon as possible.

The Correctness of the Delivery. In this paper, we are mainly concerned about the correctness of the delivery, i.e., S will not deliver a fake model if the integrity of the training has been verified, since this behavior brings no economic incentives and may affect the server's reputation potentially. To provide stronger security guarantee, here we discuss how to verify the correctness of the delivered model.

The goal of verifying the model correctness is to ensure that the delivery has the claimed prediction accuracy, S will first construct a circuit, which takes the delivery as input, to make predictions on a small test set. Then, S extends this circuit to commit the hash of the model rather than its identifier, and produces the corresponding proof. Using Merkle hash tree and the FairSwap protocol can help save the computation overhead of this step. If this proof can pass the verification algorithm, the correctness of the delivered model is verified. For the prediction service, this step is more effective since the prediction results may have smaller size and can be hashed by the circuit directly.

8 Performance Evaluation

In this section, we present the implementation and experimental results to show the performance of VeriML.

8.1 Implementations

Setup. Our system is implemented in Java. We use the jsnark compiler [56] to produce the circuit and adopt libsnark [43] as the backend. The server side which executes the proving part runs on a desktop with Ubuntu 16.04, Intel Xeon W-2133 CPU and 64 GB RAM. The client side runs on a laptop with Ubuntu 16.04, Intel Core i5-4460S CPU and 16 GB RAM to execute the generation and verification.

Datasets. The datasets we use are as follows. The Boston House Price dataset (BHP) [57] contains 506 samples and 13 features, and the label is the house price. The banknote authentication (BA) dataset is extracted from images of genuine and forged banknote-like specimens with 4 features and a binary label [58]. The Nursery dataset [59] has 12,960 samples and 8 features (we bucket the features to 27 bins), and the label is the outcome of Slovenian nursery admission process. The dataset US [60] contains 600 K census records with 15 features from the United States. MNIST has 60K images of handwritten digits, each with 784 features [61].

In VeriML, we implemented six popular machine learning algorithms: linear regression, logistic regression, neural network, support vector machine, K-Means and decision tree. Concretely, we implemented decision tree based on the CART algorithm [62], and we implemented a 4-layer fully-connected neural network, which is the same as [8], [25] with two hidden layers (each hidden layer has 128 neurons) and the approximated sigmoid activation function. The loss function is the cross entropy function. All the other algorithms are implemented based on the standard versions.

From an effiency point of view, we calculated the running time and communication overhead by tuning the batch size, the main parameter that can affect the performance. As a comparison, we evaluate the time of the native execution by running the constructed circuit directly in jsnark, i.e., the time of only executing the essential computations. We do not choose to compare with existing ML packages, because the computation processes without fixed-point number representations are incompatible with VC techniques, and the plenty of optimization methods in these packages have not been implemented by circuits yet. Furthermore, we evaluate the effect of our sampling and checkpoint strategy. From the accuracy point of view, we evaluated the effect of fixed-

TABLE 1
Verification Costs of All Implemented Algorithms

Algorithm	Dataset	Dimension	Batch	KeyGen (s)	Prove (s)	Verify (ms)	Native (ms)	EK (MB)	VK (MB)
Linear Regression	BHP	13	32	2.0	0.46	5.9	32	11.7	18.6
			64	2.0	0.47	6.4	34	11.8	36.1
			128	2.1	0.51	7.5	36	12.1	70.9
Logistic Regression	US	15	32	2.1	0.42	6.1	33	11.9	21.1
			64	2.2	0.45	6.2	34	12.1	41.0
			128	2.2	0.47	7.6	37	12.5	80.1
NN	MNIST	784	32	25.4	12.0	989	475	159.1	19.5
			64	27.0	13.4	1068	825	166.7	21.2
			128	33.4	16.7	1231	1523	185.9	24.4
SVM	US	15	32	2.1	0.43	6.1	94	11.6	24.2
			64	6.4	0.45	6.5	103	11.7	45.4
			128	2.2	0.47	7.5	115	11.8	87.8
K-Means	MNIST	784	32	8.7	2.9	42.3	334	52.4	1.6
			64	15.3	5.4	64.5	601	99.4	2.5
			128	27.3	9.6	106	1253	189.5	4.5
Decision Tree	Nursery	27	–	6.2	1.8	5.5	136	11.9	0.01

point approximation and compared all the proposed approximated activation functions with benchmarks and existing methods on 4 real-world datasets.

8.2 Computation Overhead

Since our design is based on sampling which randomly chooses multiple iterations for verification, we first present the impact of the sampling strategy, and then discuss the usability of our system in practice. We assume that both the server and the client store the dataset and compile the circuits in advance.

Performance of Proving and Verification. Table 1 shows the overheads of each verification in all the implemented algorithms. Obviously, to make the ML service available in practice, the system needs to ensure that the verification overhead on the client side is less than executing the training locally. If the claimed number of iterations is N, the interval length is m, the number of challenges is c, the running time of KeyGen is t_k, and the proving time, verifying and executing one iteration are t_p, t_v and t_e, respectively. Since the overhead of executing KeyGen is one-off for each task, we have $t_k + ct_v < Nt_e$. According to Table 1, VeriML is practicable when the task has hundreds of iterations.

The overhead only depends on the size of the quadratic program, i.e., the number of inputs and needed multiplications. As the overhead increases linearly with this size, the performance can be estimated according to the dataset, learning algorithm, and a benchmark. For example, the overhead of running linear regression on a 100-dimension dataset is twice that of the 50-dimension dataset. Therefore, the batch size is a major factor affecting the overhead. Table 1 shows that the overhead grows linearly with the batch size. Note that training decision tree does not require to set the batch size, so we omit the discussion here.

Efficiency of the Sampling Strategy. On the server side, to make the system as economical as possible, we also expect to prove that the overhead is less than executing the training locally. So we have $\frac{m}{2}ct_e + ct_p < Nt_e$, i.e., the total time of generating proofs and retrieving the model's states is less than that of training. For the implemented algorithms, VeriML is economic when the task has thousands of iterations.

The latency is approximately equivalent to the total time of retrieving states, producing and verifying proofs, which can be written as $\frac{m}{2}ct_e + c(t_p + t_v)$. Taking the linear regression task as an example, we can see from Table 1 that the time of retrieving one state is estimated to be 0.9s, while setting the interval length is 50. The whole checkpointing scheme requires about 11.7s with 13 challenges. The time costs of proving and verification are 6.5s and 0.1s, respectively. Furthermore, the server has additional storage costs in retrieving the model's state from checkpoints. If the bit length of parameters is l, and the model's dimension is d, the storage cost can be calculated as ldN/m bits. If the total number of iterations is 10K, the storage cost is 10.2KB. Fig. 5 shows the effects of interval length on the storage overhead and retrieving time. We can see that a bit length of 50 is an appropriate balance of computation and storage costs.

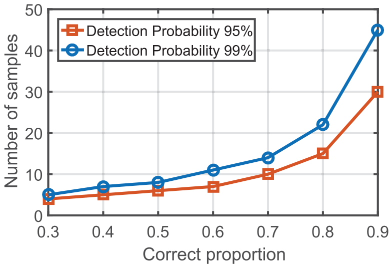


Fig. 4. Sampling strategy.

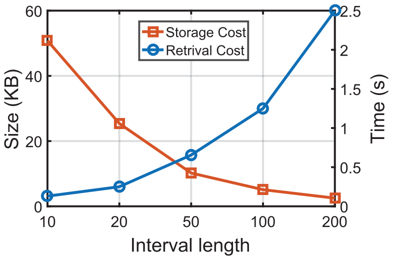


Fig. 5. Impact of interval length.

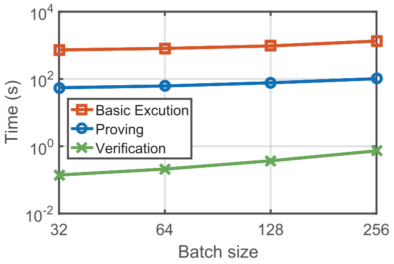


Fig. 6. Run-time of verifying linear regression.

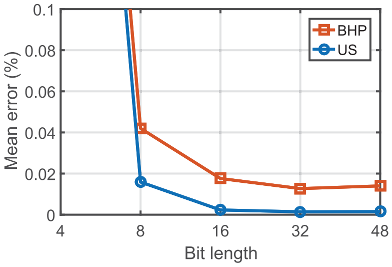


Fig. 7. Impact of bit length.

Fig. 6 shows the runtime of verifying the whole training process using linear regression as a concrete example. Here we assume that the claimed number of iterations is 100K, the dimension is 13, the batch size is 128, the interval length is 50, and the proportion of genuine iterations is 70 percent. The server can prove the claimed workload by spending about 2.2 percent of the native execution time, and the cost of verification by the client is far less than the re-execution of the training task.

Choosing the Number of Samples. As the model's structure is known by the service provider, cheating is costless by generating random model parameters. Thus, the service provider can lie about the resource consumption arbitrarily. If the user cannot verify the bill, the service provider's reputation will not be damaged because no evidence about cheating can be found from the results. Even if the user has doubts on the too long training time or low accuracy, the service provider can ascribe this to the inappropriate ML algorithm or low-quality data. Nevertheless, such a malicious behavior can be detected by our scheme with high probability. If the service provider expects to earn 10 percent additional benefits, this will be detected by verifying 29 iterations with a probability higher than 95 percent, and the verification of only 16 iterations can detect this misbehavior with a probability higher than 81 percent. Therefore, lying about the workload brings a huge risk, and the evidence of this behavior will significantly damage the service provider's reputation.

Since the proportion of forged iterations is unknown to the client, the number of sampled iterations is determined by the client's expected confidence. Fig. 4 presents the needed samples for different proportions of the correct-executed iterations. The x-axis is the expected correct proportion of all iterations, and the y-axis is the number of challenges. With the increasing proportion of genuine iterations, the required number of challenges will rapidly increase and cause high latency.

The right-hand side of Eq. (4) shows the probability that all the sampled iterations are genuine is $\prod_{i=1}^{c}(tN+i-c)/(N+i-c)$. Obviously, the upper bound is t^{c}, which is unrelated to N. Thus, this probability is still small even if N is small.

Generally speaking, VeriML can verify the integrity of the training process efficiently with a much lower computation overhead than that needed for rerunning the program. This mainly contributes to the sampling strategy which only verifies a small partion of steps and the optimization methods specially designed for converting ML algorithms to circuits.

8.3 Communication Overhead

The communication overhead lies in transmitting the proofs and the evaluation keys.

For the first part, since the sizes of each proof and each identifier are constant (288 bytes and 256 bits, respectively), the communication overhead is only related to the number of iterations and samples, and will be affected by the kind of the algorithm. For example, with 10 K iterations and 15 proofs, it only costs about 317 KB. The communication cost of the prediction service is constant because only one proof is required, i.e., 288 bytes without the delivery of prediction results.

For the second part, the size of EK_F is linear with the number of inputs and multiplications of the circuit. Table 1 shows the sizes of the keys for different algorithms. The size of EK_F may be much larger than those of commitments and proofs. Fortunately, it can be reused for all samples in each task and transmitted in parallel during the training. Therefore, it will not affect the performance of verification.

Furthermore, the cost of executing a smart contract to exchange the key is also cheap. In Ethreum, evaluating one hash function spends 27,265 gas, which can be translated to only 0.000214 Ether or 0.047 USD for an exchange rate of 220 USD/Ether.

Hence, the one-time key generation and one-round interaction of transmitting succinct proofs mean that the communication overhead of VeriML is very low. Hence, the latency will not be affected by the network bandwidth.

8.4 Accuracy Loss

The accuracy loss in our system can be analyzed from two aspects: the fixed-point representation of rational numbers and approximate functions.

In Fig. 7, we take linear regression as an example to show the impact of the fixed-point representation. The learning rate is set as 0.05, and the number of iterations is 10K. While varying the bit length between 4 and 48, the mean error of the trained model decreases rapidly. We can observe that a 32-bit representation has no obvious accuracy loss. In our implementations, linear regression, SVM and K-Means do not include the approximate functions, so their accuracy losses can be ignored. Furthermore, the prior work [49] shows that using histogram will not affect the accuracies of decision trees.

For implementing logistic regression and neural network, we introduce the approximated sigmoid function as the activation function. Table 2 shows that the Remez method has better performance than the Taylor extension and piecewise methods.

TABLE 2
Accuracy of Approximated Sigmoid

Dataset	Sigmoid	Taylor	Remez	Piecewise
BN	**73.41**	72.60	**73.24**	73.29
US	**87.81**	85.11	**86.17**	85.84
MNIST	**95.49**	87.82	**95.58**	96.15

Rectified Linear Unit (ReLU) $f(x) = \max(0, x)$ is another popular activation function in neural networks. Applying ReLU in arithmetic circuits also incurs a huge computation cost similar to sigmoid functions. If one layer uses ReLU as the activation function, it needs to execute bn_i comparisons, which are even more expensive than verifying the matrix multiplication. Thus, the square function $f(x) = x^2$ might be suitable for replacing ReLU in VeriML [63]. Prior results show that using the square function to replace ReLU can achieve satisfactory performance (99 percent accuracy) [6], [63]. Moreover, we have evaluated a 2-degree polynomial approximation of ReLU: $f(x) = x^2 + x$ [64], which can achieve 97.79 percent accuracy on the MNIST dataset. This also confirms that using low-degree polynomials to replace the activation function is feasible.

These results show that VeriML will not cause significant accuracy loss in both linear operations and non-linear operations by choosing appropriate approximations.

9 Conclusion

In this paper, we presented the design, implementation, and evaluation of VeriML, a framework that can verify the integrity of executing machine learning algorithms of MLaaS. We designed a new commit-and-prove protocol to make commitments to the intermediate results and presented how to transform typical ML algorithms to arithmetic circuits to generate proofs. By sampling a few commitments, misbehaviors in the training process would be detected with high probability. Finally, we used a compact smart contract to achieve fair payment. Experimental results of verifying six typical ML algorithms validate that the computation and communication costs of VeriML are reasonable.

We consider two future directions. First, the efficiency of existing zero-knowledge proof schemes can be further improved. Second, seeking for better low-degree polynomial activation functions with higher accuracy is helpful in building more complex models.

Acknowledgments

The work of Qian Wang was supported in part by the National Key Research and Development Program of China under Grant 2020AAA0107700, and the NSFC under Grant U20B2049 and Grant 61822207. The work of Cong Wang was supported in part by the Research Grants Council of Hong Kong under Grant CityU 11217819 and Grant CityU 11217620, by the Innovation and Technology Commission of Hong Kong under ITF Project ITS/145/19, and by the NSFC under Grant 61572412. The work of Qi Li was supported in part by the NSFC under Grant 61572278 and BNRist under Grant BNR2020RC01013. The work of Chao Shen was supported in part by the NSFC under Grants 61822309, 61773310 and U1736205.

References

[1] AWS Billing Error Overcharges Cloud Customers, [Online]. Available: https://www.crn.com/news/cloud/aws-billing-error-overcharges-cloud-customers?itc=refresh
[2] This AI Startup Claims to Automate App Making But Actually Just Uses Humans. Accessed: Aug. 2019. [Online]. Available: https://www.theverge.com/2019/8/14/20805676/engineer-ai-artificial-intelligence-startup-app-development-outsourcing-humans
[3] Human-Guided Burrito Bots Raise Questions About the Future of Robo-Delivery. Accessed: Jun. 2019. [Online]. Available: https://thehustle.co/kiwibots-autonomous-food-delivery/
[4] Amazon Web Services (AWS). Accessed: Jul. 2020. [Online]. Available: https://aws.amazon.com/
[5] Microsoft Azure. Accessed: Jul. 2020. [Online]. Available: https://azure.microsoft.com/
[6] Z. Ghodsi, T. Gu, and S. Garg, "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4672–4681.
[7] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 619–631.
[8] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 19–38.
[9] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for C: Verifying program executions succinctly and in zero knowledge," in *Proc. Annu. Cryptol. Conf.*, 2013, pp. 90–108.
[10] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proc. IEEE Symp. Secur. Privacy*, 2013, pp. 238–252.
[11] D. Papadopoulos, C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Practical authenticated pattern matching with optimal proof size," *Proc. VLDB Endowment*, vol. 8, no. 7, pp. 750–761, 2015.
[12] Y. Zhang, C. Papamanthou, and J. Katz, "Alitheia: Towards practical verifiable graph processing," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 856–867.
[13] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," in *Proc. Annu. Symp. Foundations Comput. Sci.*, 1990, pp. 2–10.
[14] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Proc. Annu. Cryptology Conf.*, 2013, pp. 71–89.
[15] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 863–880.
[16] R. Cramer and I. Damgård, "Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free?," in *Proc. Annu. Int. Cryptology Conf.*, 1998, pp. 424–441.
[17] A. Chiesa, M. A. Forbes, and N. Spooner, "A zero knowledge sumcheck and its applications," 2017, *arXiv: 1704.02086*.
[18] R. S. Wahby, S. T. Setty, Z. Ren, A. J. Blumberg, and M. Walfish, "Efficient ram and control flow in verifiable outsourced computation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015.
[19] C. Costello *et al.* "Geppetto: Versatile verifiable computation," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 253–270.
[20] A. Kosba, C. Papamanthou, and E. Shi, "xJsnark: A framework for efficient verifiable computation," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 543–560.
[21] S. Hu, C. Cai, Q. Wang, C. Wang, Z. Wang, and D. Ye, "Augmenting encrypted search: A decentralized service realization with enforced execution," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2019.2 957 091, 2019.
[22] TrueBit - A Scalable Verification Solution for Blockchains. Accessed: Jul. 2020. [Online]. Available: https://truebit.io/
[23] O. Ohrimenko *et al.* "Oblivious multi-party machine learning on trusted processors," in *Proc. USENIX Secur.*, 2016, pp. 619–636.
[24] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," 2018, *arXiv: 1806.03287*.
[25] P. Mohassel and P. Rindal, "ABY 3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.

[26] M. Li, S. S. M. Chow, S. Hu, Y. Yan, C. Shen, and Q. Wang, "Optimizing privacy-preserving outsourced convolutional neural network predictions," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2020.3 029 899.

[27] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," *Proc. Privacy Enhancing Technol. Symp.*, 2019, pp. 26–49.

[28] J. Chen *et al.*, "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 919–933, Apr. 2017.

[29] J. Chen and P. Yu, "A domain adaptive density clustering algorithm for data with varying density distribution," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2019.2 954 133.

[30] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without PCPs," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2013, pp. 626–645.

[31] P. D. Azar and S. Micali, "Rational proofs," in *Proc. 44th Annu. ACM Symp. Theory Comput.*, 2012, pp. 1017–1028.

[32] P. D. Azar and S. Micali, "Super-efficient rational proofs," in *Proc. 14th ACM Conf. Electron. Commerce*, 2013, pp. 29–30.

[33] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya, "Incentivizing outsourced computation," in *Proc. 3rd Int. Workshop Economics Networked Syst.*, 2008, pp. 85–90.

[34] M. Campanelli and R. Gennaro, "Efficient rational proofs for space bounded computations," in *Proc. Int. Conf. Decis. Game Theory Secur.*, 2017, pp. 53–73.

[35] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1322–1333.

[36] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. USENIX Conf. Secur. Symp.*, 2016, pp. 601–618.

[37] M. Song *et al.*, "Analyzing user-level privacy attack against federated learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2430–2444, Oct. 2020.

[38] Q. Wang, B. Zheng, Q. Li, C. Shen, and Z. Ba, "Towards query-efficient adversarial attacks against automatic speech recognition systems," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 896–908, 2020.

[39] G. Maxwell, "Zero knowledge contingent payment," 2011. [Online]. Available: https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment(visited on 05/01/2016)

[40] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 229–243.

[41] G. Fuchsbauer, "Subversion-zero-knowledge snarks," in *Proc. IACR Int. Workshop Public Key Cryptography*, 2018, pp. 315–347.

[42] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 967–984.

[43] libsnark. Accessed: Feb. 2018. [Online]. Available: https://github.com/scipr-lab/libsnark

[44] R. Cleve, "Limits on the security of coin flips when half the processors are faulty," in *Proc. 18th Annu. ACM Symp. Theory Comput.*, 1986, pp. 364–369.

[45] C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio, "Noisy activation functions," in *Proc. 33rd Int. Conf. Int. Conf. Mach. Learn.*, 2016, pp. 3059–3068.

[46] J. H. Cheon, J. Jeong, J. Lee, and K. Lee, "Privacy-preserving computations of predictive medical models with minimax approximation and non-adjacent form," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2017, pp. 53–74.

[47] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "Pegasos: Primal estimated sub-gradient solver for SVM," *Math. Program.*, vol. 127, no. 1, pp. 3–30, 2011.

[48] D. Sculley, "Web-scale k-means clustering," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 1177–1178.

[49] G. Ke *et al.* "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 3146–3154.

[50] R. Motwani and P. Raghavan, *Randomized Algorithms*. London, U.K.: Chapman & Hall/CRC, 2010.

[51] L. Zhao *et al.* "Shielding collaborative learning: Mitigating poisoning attacks through client-side detection," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2020.2 986 205

[52] L. Zhao, Q. Wang, Q. Zou, Y. Zhang, and Y. Chen, "Privacy-preserving collaborative deep learning with unreliable participants," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1486–1500, 2020.

[53] R. S. Wahby, I. Tzialla, J. Thaler, and M. Walfish, "Doubly-efficient zksnarks without trusted setup," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 926–943.

[54] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 315–334.

[55] Y. Lindell and J. Katz, *Introduction to Modern Cryptography*. London, U.K.: Chapman and Hall/CRC, 2014.

[56] jsnark: A java library for building snarks. Accessed: Apr. 2019. [Online]. Available: oblivm.com/jsnark

[57] Housings. Accessed: Jul. 1993. [Online]. Available: https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

[58] Banknote. Accessed: Apr. 2013. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/banknote+authentication

[59] Nursery. Accessed: Jun. 1997. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Nursery

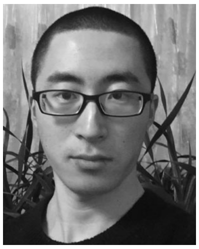
[60] Minnesota Population Center, "Integrated public use microdata series-international: Version 5.0," 2009, [Online]. Available: https://international.ipums.org

[61] MNIST. Accessed: Nov. 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[62] L. Breiman, *Classification and Regression Trees*. Evanston, IL, USA: Routledge, 2017.

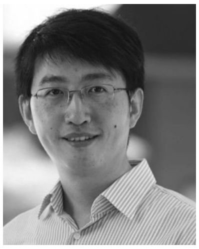
[63] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 201–210.

[64] R. E. Ali, J. So, and A. S. Avestimehr, "On polynomial approximations for privacy-preserving and verifiable relu networks," 2020, *arXiv: 2011.05530*.



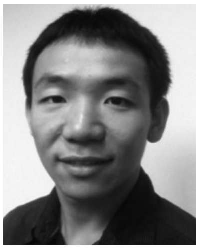
Lingchen Zhao received the BS degree from the College of Information Science and Engineering, Central South University, China, in 2016. He is currently working toward the PhD degree at the School of Cyber Science and Engineering, Wuhan University, China. His research interests include applied cryptography and secure computing.

Qian Wang (Senior Member, IEEE) received the PhD degree from the Illinois Institute of Technology, Chicago, IL. He is a professor with the School of Cyber Science and Engineering, Wuhan University. His research interests include AI security, data storage, search and computation outsourcing security and privacy, wireless systems security, big data security and privacy, and applied cryptography etc. He received National Science Fund for Excellent Young Scholars of China in 2018. He is also an expert under National "1000 Young Talents Program" of China. He is a recipient of the 2018 IEEE TCSC Award for Excellence in Scalable Computing (Early Career Researcher), and the 2016 IEEE Asia-Pacific Outstanding Young Researcher Award. He is also a co-recipient of several Best Paper and best student paper awards from IEEE DSC'19, IEEE ICDCS'17, IEEE TrustCom'16, WAIM'14, and IEEE ICNP'11 etc. He serves as associate editors for the *IEEE Transactions on Dependable and Secure Computing* (TDSC), the *IEEE Transactions on Information Forensics and Security* (TIFS) and the *IEEE Internet of Things Journal* (IoT-J). He is a member of the ACM.



Cong Wang (Fellow, IEEE) is an associate professor with the Department of Computer Science, City University of Hong Kong. His current research interests include data and network security, blockchain and decentralized applications, and privacy-enhancing technologies. He is one of the founding members of the Young Academy of Sciences of Hong Kong. He received the Outstanding Researcher Award (junior faculty) in 2019, the Outstanding Supervisor Award in 2017 and the president's awards in 2019 and 2016, all from City University of Hong Kong. He is a co-recipient of the IEEE INFOCOM Test of Time Paper Award 2020, Best Paper Award of IEEE ICDCS 2020, Best Student Paper Award of IEEE ICDCS 2017, and the Best Paper Award of IEEE ICPADS 2018 and MSN 2015. His research has been supported by multiple government research fund agencies, including National Natural Science Foundation of China, Hong Kong Research Grants Council, and Hong Kong Innovation and Technology Commission. He serveshas served as associate editor for the *IEEE Transactions on Dependable and Secure Computing*, the *IEEE Internet of Things Journal* and the *IEEE Networking Letters*, and the *Journal of Blockchain Research*, and TPC co-chairs for a number of IEEE conferencesworkshops. He is a member of ACM.

Qi Li (Senior Member, IEEE) received the PhD degree from Tsinghua University. Now he is an associate professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. He has ever worked with ETH Zurich and the University of Texas at San Antonio. His research interests include network and system security, particularly in Internet and cloud security, mobile security, and big data security. He is currently an editorial board member of the *IEEE Transactions on Dependable and Secure Computing* and ACM DTRAP.



Chao Shen is currently a professor with the School of Electronic and Information Engineering, Xi'an Jiaotong University of China. He serves as an associate dean of School of Cyber Security of Xi'an Jiaotong University. He is also with the Ministry of Education Key Lab for Intelligent Networks and Network Security. He was a research scholar in Carnegie Mellon University from 2011 to 2013. His research interests include network security, human computer interaction, insider detection, and behavioral biometrics.

Bo Feng received the bachelor's degree in computer science from Wuhan University, China, in 2015. He is working toward the PhD degree in computer science at Northeastern University, Boston, MA. His research interests include IoT security, system security, and applied cryptography.