TEE-based Selective Testing of Local Workers in Federated Learning Systems

Wensheng Zhang and Trent Muhr
Computer Science Department, Iowa State University
Ames, Iowa, USA 50011
E-mail:{wzhang,muhr}@iastate.edu

2021 18th International Conference on Privacy, Security and Trust (PST) | 978-1-6654-0184-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/PST52912.2021.9647773

***Abstract*—This paper considers a federated learning system consisting of a central aggregation server and multiple distributed local workers, all having access to trusted execution environments (TEEs). For the local workers, which are untrusted but economically-rational, to conduct local learning honestly, we propose a TEE-based selective testing scheme that also combines techniques from applied cryptography, game theory and smart contract. Theoretical analysis of the scheme indicates that only a small number of tests are needed to enforce honest execution by the local workers. Implementation-based experiments compare the cost of the proposed scheme against two reference schemes (i.e., the original scheme without security measure and the all-SGX scheme which conducts training completely in an SGX enclave). The results show that, our proposed scheme incurs much lower cost at the SGX enclave though introducing a higher cost at the untrusted execution environment. We argue that this tradeoff is appropriate given that computing in the untrusted environment can access more resources and is cheaper than in the trusted environment. The experiment results also show that, the increase of the cost in the untrusted execution environment get smaller as the size of the training model increases, which demonstrates the scalability of the scheme.**

I. Introduction

Along with the increasing popularity of federated learning [1], comes a host of challenges, including communication efficiency, non-I.I.D. distribution of data samples, dynamic participation, and security and privacy.

Related Works: The distributed nature of federated learning introduces new security and privacy concerns. For example, a curious server could infer information about the data used by local workers. Making use of cryptographic primitives such as masking and public key cryptography, secure aggregation [2], [3], [4], [5] has been an attempt to prevent this. Though a lot of research has been conducted to address the security and privacy risks due to possibly misbehaving aggregation server, it is imperative to also secure the system against misbehaving local workers. Without proper security measure in place, a local worker may deviate from honest behavior in various ways. For instance, it may use faked rather than truthful data in local training; not select its local training data as randomly as expected; not use as many as expected data samples; not honestly execute the local training operations. Such misbehavior might be treated as data poisoning attack. However, countermeasures to such attack [6], [7], [8], [9] may not be sufficiently accurate or timely, and not be able to quickly identify and thus avoid and/or punish the misbehaving local workers. Particularly, when certain privacy protection mechanisms are applied at the server as well, the inputs from different local workers could be blindly aggregated which makes it even more challenging to detecting misbehavior and identify misbehaving local workers. Besides, there has also many research [10] on secure and private outsourcing of deep neural network based inference, based on differential privacy [11], homomorphic encryption [12], [13], [14], or TEE [15]. However, the schemes proposed may not be directly applicable to address misbehaving local workers in federated learning.

Our Work: We propose a new scheme to detect misbehavior directly and immediately at the local workers in a trusted manner. The scheme is based on the following ideas.

First, contemporary servers, PCs and mobile devices are commonly equipped with TEEs based on Intel SGX [16], TrustZone [17], etc. To directly monitor the behavior of local workers, monitoring function can be deployed to these TEEs.

Second, the monitoring function can be implemented by directly and immediately repeating a selected subset of the operations the local workers are expected to have conducted. However, the execution in TEEs is less efficient than in untrusted environments due to limited resource. Hence, we propose a game-theoretic design to minimize the involvement of TEEs while still ensuring the reliability of monitoring. Specifically, our design requires each economically-rational local worker to make a deposit of a small amount (e.g., the cost for executing only one stage of training a neural network) when it joins the federated learning system; as we prove, testing the correctness of only a small number (e.g., two) of the operations per stage that the worker is expected to conduct can enforce it to behaves honestly.

Third, directly testing even a single operation, e.g., the computation of one output of a convolutional or fully-connected layer, could still be very inefficient because a large number of relevant data items might be involved. Hence, we further propose to convert heavy tests into lightweight tests to make the scheme even more efficient and practical. Due to page limit, the optimization is elaborated in [18].

We implement the proposed scheme, and evaluate its performance for forward/backward propagation through convolutional/fully-connected layers. We also compare our scheme with two reference schemes: the original scheme which runs completely in untrusted execution environment without any security measure in place; the all-SGX scheme

which runs completely in an SGX enclave and thus ensures honest execution. The performance is measured by the running time in the SGX enclave and in the untrusted environment. According to the evaluation, our scheme only incurs a small testing cost in the TEE, though it introduces extra operations (such as constructing Merkle hash trees) to be conducted in the untrusted execution environment. However, such overhead is found to be comparable to the costs of the original and all-SGX schemes, and it gets smaller as the input/output size increases. Also note that, the total time that our scheme spends is similar to that by the all-SGX scheme, but the majority of our scheme's time is spent in the untrusted environment, which is easy to be reduced through parallelism. This is different for the all-SGX scheme, for which the execution time is all spent in the SGX enclave and thus more difficult to reduce.

Organization: In the rest of the paper, Section II introduces background and problem description. Section III describes and analyzes the basic framework of our proposed scheme. Implementation-based evaluations are presented in Section IV. Finally, Section v concludes the paper.

II. Problem Description

We consider a system composed of a central server and multiple local workers. Each local worker has both trusted execution environment (TEE) and untrusted rich execution environment (REE). Each local worker has its own training data, which should never be exposed to others. Coordinated by the central server, the local workers collaborate in building a global neural network model. The central server has an initial model; then, the system works round by round to update it. In the beginning of each round, each local worker downloads the current global model from the central server, and uses its own data to update the weights of connections. The central server collects the updates, and applies them to the global model to get a newer version used in the next round.

Assumptions for Local Training Data: Each local worker's training data is represented as records. We assume that the validity of each record can be verified based on a digital signature mechanism. For example, it is reasonable to assume that valid medical data records should be digitally signed by certain authorized personnel and the digital signatures can be verified using certain certified public keys, so that any user knowing the certified public keys can verify such signatures and thus trust the information carried by the signed records. Hence, each record is assumed to bear the following format: $\langle x_1, \cdots, x_{n_X}; y_1, \cdots, y_{n_Y}; \sigma \rangle$. Here, $(x_1, \cdots, x_{n_X})$ is the vector of n_X input features, $(y_1, \cdots, y_{n_Y})$ is the tag vector of n_Y elements, and σ is a digital signature. A Merkle hash tree for the record is built with the hashes of $x_{i,1}, \cdots, x_{i,n_X}$ and $y_{i,1}, \cdots, y_{i,n_Y}$ as leaf nodes, the root of the above hash tree is called *record hash*, and the hash is signed with an authorized private key to obtain a verifiable digital signature.

Security Assumptions and Goals: We aim to address the following attacks that may be launched by a misbehaving local worker: using invalid data for local learning; failing to choose training data randomly; failing to honestly conduct computation. We assume local workers could be selfish or lazy, by pretending to have more data for training than they actually have, or by faking (skipping the complete procedure of) computation to save cost. Hence, we model them as economically-rational; that is, they always intend to maximize their profits, computed as the incomes minus the costs that they have to pay.

A variety of side-channel attacks have been discovered for SGX-based designs, which are out of the scope of this paper. Note that, our proposed scheme executes testing only after the untrusted local worker has completed the tested tasks and submitted commitments, which cannot be changed. Hence, even if the worker can observe (but cannot manipulate) the execution of an enclave.

III. The TEE-based Selective Testing Scheme

A. Primitives: Commitment and Verification

We introduce primitives *Construct_Commit* and *Verify_Element*. Primitive *Construct_Commit* takes a vector $\vec{v}$ as input, constructs a Merkle hash tree $MT(\vec{v})$ with the hashes of each elements of $\vec{v}$ as leaf nodes. Then, the root of the tree is returned as the *commitment* of the vector. Meanwhile, for each element $\vec{v}[i]$, the sequence of its corresponding co-path hash values on $MT(\vec{v})$ is returned as the evidence for verifying it as the i-th element of $\vec{v}$. Accordingly, primitive *Verify_Element* takes four arguments, i.e., an element u, an index i, the commitment $comm(\vec{v})$ for certain $\vec{v}$ and an evidence $evid$. It assumes u as the i-th element of $\vec{v}$ and makes use of the assumed co-path hash values in $evid$ to recompute the root of $MT(\vec{v})$. If and only if the recomputed root is the same as $comm(\vec{v})$, the element u is confirmed.

B. SIMD Computation

When training a neural network model, on each stage of each layer, the same type of operation needs to be performed over different data. Taking the forward propagation over a convolutional layer l as example, there are two stages. For the first stage (transformation), the outputs of layer $l-1$ are transformed to the inputs of layer l as follows: the input of each neuron at layer l is computed as the inner product of a filter matrix and a set of output elements of layer $l-1$. For the second stage (activation), at each neuron of layer l, the input is fed to an activation function to obtain the output of the neuron. We call such computation paradigm at each stage of each layer as single instruction multiple data (*SIMD*) computation, and formalize it as $\vec{Y} = g(\vec{X})$, where $g(\cdot)$ represents the operation, $\vec{X}$ the vector of input and $\vec{Y}$ the vector of corresponding output. When instantiated for the aforementioned transformation stage of convolutional layer l, g stands for the inner product operation, $\vec{Y}$ is the vector of input to layer l, and $\vec{X}$ is the vector containing all the subsets of layer $l-1$'s output elements used to compute the elements in $\vec{Y}$. Therefore, the whole model training procedure can be formalized as a sequence of SIMD computations.

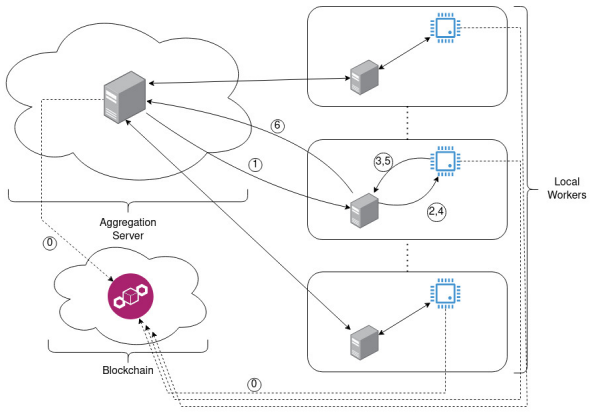

Figure 1: System overview: The system is comprised of an aggregation server and some number of untrusted workers, all with access to a blockchain with smart contract capabilities. (0) Each party participates in a smart contract. (1) The endorsed model is downloaded from the server. (2) Worker has the local enclave validate records for training and asks the enclave to select one for training. (3) TLM sends record choice to untrusted worker. (4,5) The untrusted worker and its local enclave engage in selective testing. (6) The final update is sent to the aggregation server, endorsed by the worker's enclave.

C. The Proposed Selective Testing Scheme

1) System Components: We define the following system components: a central server (CS), multiple untrusted local workers (UW), and one trusted local monitor (TLM) co-residing with each UW. Here, each TLM is run in a TEE. When a UW joins the system, the TLM co-located with the UW should authenticate itself to the CS. Then, the TLM should set up secret pairwise keys with the UW and the CS, respectively, to secure their communications. Also, we assume the CS does not collude with any UW.

2) Signing Smart Contract: The CS signs a smart contract with each UW. With the contract, the UW makes a *small* deposit d that is only required to be larger than twice of the maximal cost of executing one stage of SIMD computations. If the UW is found dishonest by its co-located TLM through selective testing, its deposit will be taken by the CS and it will be evicted from the system; otherwise, the UW will remain in the system and continue its participation.

3) Validating and Preparing Local Data Records: After a UW has signed the above smart contract with the CS, it requests its co-located TLM to validate its data records and prepare them for federated learning. Each record $\langle\ x_1, \cdots, x_{n_X}, y_1, \cdots, y_{n_Y}, \sigma\ \rangle$ is processed as follows. First, the TLM checks the validity of the record. That is, letting $\vec{v} = (x_1, \cdots, x_{n_X}, y_1, \cdots, y_{n_Y})$, it computes $comm(\vec{v}) = Construct\_Commit(\vec{v})$ and verifies if σ is a valid signature of $comm(\vec{v})$. Second, the TLM assigns a unique identity $i \in [n_R]$ to the record, where n_R is the number of such records. Thus, each record can be denoted as $R_i = \langle i, x_{i,1}, \cdots, x_{i,n_X}, y_i, \cdots, y_{i,n_Y}, \sigma_i \rangle$. Then, a Merkle tree for all of the n_R records is built with $h_i = hash(i|\sigma_i)$ for all $i \in [n_R]$ as leaf nodes. The root hash of the tree is denoted as h_R. The UW keeps this Merkle tree for later use, but the TLM only keeps h_R and n_R.

4) Initializing Each Round (i.e., testing for layer 1): After its local data records have been validated and prepared for federated learning by its co-located TLM, a UW can formally participate the federated learning round by round.

The UW downloads the current global model from the CS. The model should be signed by the CS so that a malicious UW cannot modify them before giving it to the TLM. For the simplicity of presentation, we assume that only one record is processed in each round. The TLM randomly selects an ID $i \in [n_R]$ at the beginning of a round, and asks the UW to pick the record with the ID for training. In response, the UW retrieves the content of the selected record (i.e., R_i), the record hash (i.e., $h_i = hash(i|\sigma_i)$), and the corresponding co-path hash values on the Merkle tree of all n_R records. Then, it communicates h_i and the corresponding co-path hash values, which are called the *evidence* of the input, to the TLM. The TLM verifies it by recomputing the root hash using h_i and the evidence, and checking if the recomputed root hash is the same as h_R. Once the verification succeeds, the TLM records h_i and proceeds with the rest of the round; otherwise, it identifies the UW as dishonest and quits the system.

5) Testing for Each Hidden Layer: The operations at each hidden layer include one or more stages. Along with a UW's execution at each stage, its co-locating TLM conducts selective testing for the stage. The operations of the UW and TLM can be generally modelled as follows:

Let a stage have n same-type computations, $\vec{X}$ be the input vector, $\vec{Y}$ be output vector, and $g(.)$ be the computation function. Before the stage starts, the TLM should have already obtained $comm(\vec{X})$ and the UW should be able to provide evidence for verifying each input.

To start the stage, the UW evaluates $g(.)$ with every element of $\vec{X}$ to obtain the corresponding output element in $\vec{Y}$. Then, it computes the commitment and evidences for $\vec{Y}$ by calling $Construct\_Commit(\vec{Y})$, keeps the results, and sends $comm(\vec{Y})$ to the TLM. Upon receiving $comm(\vec{Y})$, the TLM randomly selects p out of the n computations to test. For each of the selected computation $i \in [n]$, with $\vec{X}[i]$ denoting the input element that should be used in the computation and $\vec{Y}[i]$ denoting the expected output element, the testing is as follows: The TLM requests the UW for the input element (denoted as u_0) and output element (denoted as u_1) of computation i, as well as the evidences ($evid_0$ and $evid_1$ respectively) for verifying these elements to be $\vec{X}[i]$ and $\vec{Y}[i]$ respectively. Once receiving the above, the TLM calls $Verify\_Element(\ u_0, i, comm(\vec{X}), evid_0)$ and $Verify\_Element(\ u_1, i, comm(\vec{Y}), evid_1)$ to verify if $u_0 = \vec{X}[i]$ and $u_1 = \vec{Y}[i]$. Then, it checks if $g(u_0) = u_1$. If any of the tests fails, the TLM identifies UW as dishonest and stops participation.

6) Testing for Layer $\mathcal{L}$: The final layer computes the loss function during the forward propagation, and computes the gradients for its input elements (i.e., the output elements from

the last hidden layer). Since these computations are not heavy, the TLM directly repeat them.

7) Endorsing Model Updates: A TLM should endorse the model updates computed by its co-located UW as long as the UW is not found dishonest. The CS only accepts a UW's model updates that have been endorsed by its co-located TLM; a UW that fails to provide endorsed model updates is not allowed to get the current global model from the CS and thus is evicted from the federated learning system.

D. Game-theoretic Analysis of Selective Testing

We model the interactions between the CS and each UW as an infinite extensive game with perfect information, denoted as $G = (P, A, U)$. Here, $P = \{CS.TLM, UW\}$ is the set of players where $CS.TLM$ represents the coalition including CS and TLM. A is the set of actions taken by the players, including all the combinations of the n same-type computations to fake and all the combinations of the n computations to test. As we treat the n computations equally, the action set that the UW can take is denoted as $A_{uw} = \{0, 1, \cdots, n\}$ where each element represents the number of computations that the UW randomly chooses to fake; the action set that the CS can take is denoted as $A_{cs.tlm} = \{0, 1, \cdots, n\}$ where each element represents the number of computations that the CS.TLM randomly chooses to test. $U = \{U_{uw}, U_{cs.tlm}\}$ is the players' utility functions.

The UW's utility is defined as:

$$U_{uw}(A_{uw}, A_{cs.tlm}) = \begin{cases} B - (c_c(n) - c_c(A_{uw})) & \text{if not detected;} \\ -d - (c_c(n) - c_c(A_{uw})) & \text{if detected.} \end{cases} \quad (1)$$

It says that, if none of the A_{uw} faked computations is detected, the UW's utility is $B - (c_c(n) - c_c(A_{uw}))$, where B is the UW's benefit from sharing the results of federated learning (by staying in the system) and $c_c(x)$ is the cost of honestly executing all the x computations. Note that, here we assume that faking a computation does not have computation cost, thus the computation cost is $c_c(n) - c_c(A_{uw})$ when A_{uw} of the n computations are faked. If any of the A_{uw} faked computations is detected, the UW loses its deposit; hence, its utility becomes $-d - (c_c(n) - c_c(A_{uw}))$.

Similarly, the CS.TLM's utility is defined as:

$$U_{cs.tlm}(A_{uw}, A_{cs.tlm}) = \begin{cases} B' - c_t(A_{cs.tlm}) & A_{uw} = 0; \\ -c_t(A_{cs.tlm}) + d & A_{uw} > 0 \text{ and detected;} \\ -Penalty - c_t(A_{cs.tlm}) & A_{uw} > 0 \text{ and not detected.} \end{cases} \quad (2)$$

If there is no faked computation (i.e., $A_{uw} = 0$), the CS.TLM's utility is $B' - c_t(A_{cs.tlm})$ where B' is the benefit from having the UW in federated learning and $C_t(x)$ is the cost for detecting x randomly-selected computations. If there is faked computation and it is detected, the CS.TLM takes the UW's deposit and thus its utility is $-c_t(A_{cs.tlm}) + d$. If none of the A_{uw} faked computation is detected, the CS.TLM is penalized by $Penalty$ for the failure in detection and thus its utility is $-Penalty - c_t(A_{cs.tlm})$.

The goal of the CS and TLM coalition is to enforce an economically-rational UW to execute all n computations honestly. The following theorem (for which the proof is provided in [18]) states the conditions for the goal to be attained.

Theorem 1. *For an economically-greedy untrusted local worker (UW) who aims to maximize its utility, if the CS and TLM coalition's testing probability* $\frac{A_{cs.tlm}}{n} > \frac{1}{n}$ *(i.e.,* $A_{cs.tlm} > 1$*) and the UW's deposit* $d \geq \frac{c}{1-e^{-(A_{cs.tlm}-1)}}$*, where* c *is the cost for executing all the* n *computations, the UW should honestly execute all the* n *computations.*

Based on the above theorem, letting $p = 2$, the UW is only required to make a deposit of $\frac{c}{1-e^{-1}} < 2c$ and the TLM only needs to test 2 of the n operations. When applying our proposed scheme, c is the maximal cost for executing any stage of the procedure of training a neural network model, which is small in practice. Hence, our proposed scheme is practical.

IV. Performance Evaluation

We implement a prototype of our proposed scheme and also two reference schemes: *Original (No-SGX) Scheme*, with which the REE runs the convoluntional and fully-connected layer functions without any security consideration; *Full-SGX Scheme*, with which the SGX enclave runs the convolutional and fully-connected layer functions. In the full-SGX scheme, due to limited trusted memory space, data should be loaded from the untrusted memory to the enclave before being processed and the processing results should be stored back to the untrusted memory. For data integrity, a hash of the data is computed and stored securely in enclave before the data is stored to the untrusted memory; the hash is recomputed and compared to the stored hash when the data is re-loaded to the enclave. The above three schemes are evaluated on a computer with Intel Core i5-8400 CPU (2.80GHz) of six cores and a RAM of 8.00GB. The evaluation results are presented and discussed in the following.

Convolutional Layer: Table I shows the costs of the schemes for the forward propagation through a convolutional layer, as the input size varies. The original scheme's cost is denoted as *original fwd* and the full-SGX scheme's cost is *SGX fwd*. For our scheme, the cost incurred at the REE is dentoed as *new fwd* and the cost for selective test incurred at the SGX enclave is *selective test*. All the costs are measured as the computation time in micro-second.

input size	original fwd	SGX fwd	new fwd	selective test
16×16	124	145	303	35
32×32	818	865	1265	41
64×64	3990	4161	5242	59
128×128	17705	18382	21065	118
256×256	74196	76868	84815	360

Table I: Fwd Prop Cost for Convolutional Layer (unit: μs): Impact of Input Size. $stride = 2$, $filter\_size = 8 \times 8$, and $filter\_number = 16$.

According to Table I, our scheme introduces higher cost at the REE, as the price of significantly reducing the cost at the TEE. The results also demonstrate that, when the input size is not small (i.e., greater than 32×32), our scheme does not increase the cost at the REE significantly (i.e., 1.14-1.55 times of the original scheme) while incurring significantly lower cost at the TEE (i.e., 0.5%-4.7% of the full-SGX scheme).

filter #	original fwd	SGX fwd	new fwd	selective test
4	4411	4666	5275	111
8	8866	9201	10577	117
16	17626	18587	20939	111
32	35367	36730	41942	115

Table II: Fwd Prop Costs for Convolutional Layer (unit: μs): Impact of Output Size. $stride = 2$, $filter\_size = 8 \times 8$, and $input\_size = 128 \times 128$.

stride	original fwd	SGX fwd	new fwd	selective test
1	68941	71575	78447	123
2	17626	18587	20939	111
4	4583	4789	6009	110
8	1215	1297	1857	107

Table III: Fwd Prop Costs for Convolutional Layer (unit: μs): Impact of stride. $filter\_size = 8 \times 8$, $filter\_number = 16$ and $input\_size = 128 \times 128$.

filter size	original fwd	SGX fwd	new fwd	selective test
8×8	17708	18449	21117	113
16×16	61104	53713	64606	132
32×32	179681	149620	182581	149
64×64	319404	268115	320935	152

Table IV: Fwd Prop Costs for Convolutional Layer (unit: μs): Impact of filter size. $stride = 2$, $filter\_number = 16$ and $input\_size = 128 \times 128$.

Similar trends have been shown by Tables II, III, and IV, as the number/size of the filters or the stride changes. Specifically, our scheme introduces a slightly higher cost at the REE (i.e., 1.01-1.53 times of the original scheme) and incurs much lower cost at the TEE (i.e., 0.1%-8.2% of the full-SGX scheme).

Table V shows the costs of the three schemes for backward propagation through a convolutional layer, as the size of the input varies from 16×16 to 256×256. According to the table, the full-SGX scheme's cost (denoted as *SGX bwd*) is higher than (i.e., about twice of) the original scheme's cost (denoted as *original bwd*), because the full-SGX scheme needs to load and check the integrity of the inputs and outputs of the layer. Our scheme's cost at the REE (denoted as *new bwd*) is also high because the REE needs to construct large Merkle hash trees to facilitate selective testing. Specifically, when the input size is not large (i.e., 32×32 or smaller), the cost at the REE is as high as 5-24 times of the original scheme's cost. However, when the input size becomes larger than 64×64, the cost at the REE becomes only 1.5-2.3 times of the original scheme's cost. Particularly, the cost is even smaller than the full-SGX scheme's cost when the input size is 128×128 or larger. Our scheme's cost at the TEE (i.e., *selective test*) is the smallest; i.e., 6-28% of the full-SGX scheme's cost when the input is no greater than 32×32 and only 0.3-1.6% of the full-SGX scheme's cost when the input size is 64×64 or larger.

input size	original bwd	SGX bwd	new bwd	selective test
16×16	188	398	4611	112
32×32	1249	2516	6254	139
64×64	6058	12049	13768	191
128×128	26771	53107	43840	305
256×256	112449	223482	173460	658

Table V: Bwd Prop Costs for Convolutional Layer (unit: μs): Impact of Input Size. $stride = 2$, $filter\_size = 8X8$, and $filter\_number = 16$.

filter number	original bwd	SGX bwd	new bwd	selective test
4	6702	13724	11560	228
8	13426	26874	22327	253
16	26765	52957	43884	292
32	53520	105403	93123	389

Table VI: Bwd Prop Costs for Convolutional Layer (unit: μs): Impact of $filter\_number = 16$. $stride = 2$, $filter\_size = 8X8$, and $input\_size = 128 \times 128$.

stride	original bwd	SGX bwd	new bwd	selective test
1	104705	206806	130990	367
2	26765	52957	43884	292
4	6994	14308	21471	272
8	1868	4270	15292	259

Table VII: Bwd Prop Costs for Convolutional Layer (unit: μs): Impact of stride. $filter\_size = 8X8$, $filter\_number = 16$ and $input\_size = 128 \times 128$.

filter size	original bwd	SGX bwd	new bwd	selective test
8×8	26777	53085	43984	291
16×16	92652	184529	143743	342
32×32	269609	541327	447952	420
64×64	488914	981685	1037886	446

Table VIII: Bwd Prop Costs for Convolutional Layer (unit: μs): Impact of filter size. $stride = 2$, $filter\_number = 16$ and $input\_size = 128 \times 128$.

Similar trends can be observed in Tables VI, VII and VIII, where the schemes' costs are compared as the number/size of filters or the stride changes but the input size is fixed at 128×128. Specifically, our scheme's cost at the REE ranges between 1.25-3.07 times of the original scheme's, except that the cost is 8.19 times of the original scheme's when the filter size is 8×8 and stride is 8, in which case the original scheme's workload is small because the stride is large relatively to the filter size. Our scheme's cost at the TEE remains low; it ranges between 0.05-6.1% of the full-SGX scheme's cost.

Fully-connected Layer: Tables IX and X show the costs of the three schemes for the forward propagation through a fully-connected layer, as the input and output sizes vary. As shown, except for the cases when the input and output sizes are small (e.g., input size is no greater than 32 and the output size is no

input size	original fwd	SGX fwd	new fwd	selective test
32	6	114	134	33
64	11	217	148	37
128	21	410	144	36
256	43	781	201	36
512	92	1563	270	38
1024	205	3030	413	37
2048	481	6050	715	39
4096	788	12128	1196	41

Table IX: Fwd Prop Costs for Fully-connected Layer (unit: μs): Impact of Input Size. The output size is 64.

output size	original fwd	SGX fwd	new fwd	selective test
4096	191713	1600367	249100	56
2048	102136	786005	128956	56
1024	51145	383069	65157	53
512	25597	183947	33951	60
256	12820	96058	16649	55
128	4529	32506	5793	56
64	1245	12599	1443	48
32	479	6251	636	43
16	208	3092	284	43

Table X: Fwd Prop Costs for Fully-connected Layer (unit: μs): Impact of Output Size. The input size is 4096.

greater than 64), our scheme has lower cost at the REE than the full-SGX scheme. Specifically, the cost ranges between 9-69% of the full-SGX scheme's cost. Our scheme's cost at the TEE (i.e., selective test) remains the smallest. Similar trends can be seen in Tables XI and XII, which show the costs of the three schemes for the backward propagation through a fully-connected layer, as the input and output sizes vary.

input size	original bwd	SGX bwd	new bwd	selective test
32	10	198	202	33
64	20	388	393	37
128	39	752	659	36
256	74	1484	1141	41
512	151	2955	2468	45
1024	293	5824	4132	50
2048	585	11609	8305	70
4096	1178	23215	16437	102

Table XI: Bwd Prop Costs for Fully-connected Layer (unit: μs): Impact of Input Size. The output size is 64.

output size	original bwd	SGX bwd	new bwd	selective test
4096	80876	1461079	503361	152
2048	40225	730257	257843	154
1024	20426	364780	134487	150
512	10620	186273	73545	113
256	5234	93183	41460	110
128	2706	46746	24964	108
64	1444	23394	17034	108
32	701	11780	12222	106
16	298	6028	10163	103

Table XII: Backward Propagation Costs for Fully-connected Layer (unit: micro-second): Impact of Output Size. Here the input size is 4096.

V. Conclusion and Future Work

In this paper, we proposed a scheme to ensure the correctness of computations performed by an untrusted worker in a federated learning system for CNN model. Through smart contract and selectively choosing which untrusted computations to test, computational overhead performed by the TEE is minimal, drastically reduced when compared to the reference scheme with security considerations. In the future, we plan to expand the scheme to more kinds of neural networks. We will also explore the possibility to improve the performance of the commitment process by using alternative cryptographic constructions.

Acknowledgement

The work is partly supported by NSF under grant CNS-1844591.

References

[1] H. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017.

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

[3] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. Nguyen, P. Rieger, A. Sadeghi, T. Schneider, H. Yalame, and S. Zeitouni, "Safelearn: Secure aggregation for private federated learning," *2021 IEEE Security and Privacy Workshops (SPW)*, pp. 56–62, 2021.

[4] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "Verifynet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2020.

[5] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, pp. 1333–1345, 2018.

[6] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *NIPS*, 2017.

[7] D. Yin, Y. Chen, K. Ramchandran, and P. L. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," *ArXiv*, vol. abs/1803.01498, 2018.

[8] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," *ArXiv*, vol. abs/1911.11815, 2020.

[9] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *ESORICS*, 2020.

[10] F. Mirshghallah, M. Taram, P. Vepakomma, A. Singh, R. Raskar, and H. Esmaeilzadeh, "Privacy in deep learning: A survey," *ArXiv*, vol. abs/2004.12254, 2020.

[11] Z. Bu, J. Dong, Q. Long, and W. J. Su, "Deep learning with gaussian differential privacy," *Harvard data science review*, vol. 2020 23, 2020.

[12] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: applying neural networks to encrypted data with high throughput and accuracy," in *ICML 2016*, 2016.

[13] V. N. Boddeti, "Secure face matching using fully homomorphic encryption," in *BTAS*, 2018.

[14] B. Reagen, W. Choi, Y. Ko, V. T. Lee, H.-H. S. Lee, G.-Y. Wei, and D. Brooks, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 26–39, 2021.

[15] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *ArXiv*, vol. abs/1806.03287, 2019.

[16] V. Costan and S. Devadas, "Intelsgxexplained," *IACR Cryptology ePrint-Archive*, pp. 1–118, 2016.

[17] "Arm TrustZone Technology," https://developer.arm.com/ip-products/security-ip/trustzone, [Online; accessed 1-August-2021].

[18] W. Zhang and T. Muhr, "Tee-based selective testing of local workers in federated learning systems," *ArXiv*, vol. abs/2111.02662, 2021.