SafeML: A Privacy-Preserving Byzantine-Robust Framework for Distributed Machine Learning Training

Meghdad Mirabi
Technical University of Darmstadt
Darmstadt, Germany
meghdad.mirabi@cs.tu-darmstadt.de

René Klaus Nikiel
Technical University of Darmstadt
Darmstadt, Germany
rene.nikiel@gmail.com

Carsten Binnig
Technical University of Darmstadt
Darmstadt, Germany
carsten.binnig@cs.tu-darmstadt.de

***Abstract*—This paper introduces SafeML, a distributed machine learning framework that can address privacy and Byzantine robustness concerns during model training. It employs secret sharing and data masking techniques to secure all computations, while also utilizing computational redundancy and robust confirmation methods to prevent Byzantine nodes from negatively affecting model updates at each iteration of model training. The theoretical analysis and preliminary experimental results demonstrate the security and correctness of SafeML for mode training.**

***Index Terms*—Byzantine Robustness, Computational Redundancy, Machine Learning, Privacy Preserving, Secret Sharing**

I. Introduction

Machine learning technology has gained significant popularity in both academia and industry due to its impressive performance in various fields, such as face recognition [1], [2], object classification [3], [4], and object detection [5]. Despite its success, the accuracy of machine learning models heavily depends on the volume of high-quality data used for training, which can result in increased computational demands and memory requirements [6]–[9].

To meet these requirements, researchers have been exploring distributed machine learning solutions, and in response, commercial solutions such as "Machine Learning as a Service" (MLaaS) have emerged, enabling customers to outsource the storage and computation requirements of their machine learning algorithms to Cloud Service Providers (CSPs) [10]–[12]. Still, this approach raises significant privacy concerns due to the sensitive nature of the data involved. Additionally, there is a risk of cloud servers returning erroneous results, potentially influenced by financial incentives [8], [11], [13] or Byzantine attackers seeking to corrupt the computational results [14]–[17]. Research has shown that even one Byzantine node can introduce significant bias and inaccuracies in the final model [15], [18]. Therefore, it is crucial to find a unified solution that can mitigate the risk of Byzantine attacks while ensuring privacy during model training.

The machine learning training phase poses a significant risk of privacy leakage, primarily from the input data and the training model itself. The input data often contains sensitive information that needs to be protected from untrustworthy cloud servers. Similarly, the model parameters and intermediate results generated during model training should not be disclosed to unauthorized parties. Although existing works such as [19]–[24] focus on preserving the privacy of input data, recent studies [25]–[28] have demonstrated that the model parameters can be used to reverse-engineer the input data if they are not adequately protected during the training phase.

Privacy-preserving Byzantine-robust solutions currently use either cryptographic techniques such as Interactive Proof Systems [17], [29], Homomorphic Encryption [30], [31], and Zero-Knowledge Proofs [32]–[34] to verify the computation results, or hardware-assisted Trusted Execution Environment (TEE) [13], [35]. However, these solutions come with limitations. Cryptographic-based solutions require significant computation and communication resources, while TEE-based solutions, although helpful, are not entirely secure and can still be vulnerable to side-channel attacks, including cache timing [36]–[38], branch shadowing [39], and page fault attacks [40], [41].

Recently, secret sharing schemes have gained attention for designing efficient privacy-preserving machine learning frameworks [6]–[9]. However, these schemes assume an *honest-but-curious* adversary model and cannot provide resistance against *malicious adversaries* (i.e., Byzantine attacks), where the adversary can intentionally tamper with the data, provide incorrect computation results, or collude with other Byzantine nodes to gain access to sensitive information.

Taking into account the limitations of existing solutions and recognizing its significance, this paper presents SafeML, a distributed machine learning framework that addresses privacy and Byzantine robustness concerns. To secure all computations performed by computing nodes during model training, SafeML utilizes secret sharing and data masking techniques. Furthermore, it employs computational redundancy and robust confirmation methods as strong and perfect recovery measures to filter out incorrect intermediate results returned from malicious adversarial nodes. To achieve this, SafeML decomposes all operations involved in model training into simple operations such as addition and multiplication, which can be executed efficiently over secret shared data in a privacy-preserving form. SafeML also uses groups of computing nodes to perform duplicate computation, where these nodes mask their computation results and send them to a set of other computing nodes to verify their computation. To agree on a computation result, this group of computing nodes finds the highest frequency of occurrence of a value received as the correct computation result. By combining these methods, SafeML offers a robust and trustworthy solution for model training in the presence of adversarial attacks.

The main contributions of this paper are as follows:

- We propose SafeML as a distributed machine learning framework which employs a set of secret-sharing based protocols intermixed with a robust confirmation to address privacy and Byzantine robustness during model training.
- We prove through our security analysis that SafeML is highly effective in protecting against both *honest-but-curious* and *malicious* adversary models. It not only detects malicious behavior but also ensures robustness against malicious adversaries.
- We conduct preliminary experiments to demonstrate the feasibility of SafeML for distributed model training.

The rest of the paper is organized as follows: Section II presents the preliminary concepts. Section III describes the details of SafeML. Section IV provides the security analysis of SafeML. Section V presents the preliminary experimental results for model training in SafeML. Section VI compares SafeML with the current solutions for the development of distributed machine learning frameworks. Finally, Section VII concludes and discusses future directions.

II. Preliminaries

In this section, we review additive secret sharing scheme (ASS) as the fundamental technique used in SafeML.

In ASS, the *secret*, e.g., x, is split into n different *additive shares* $[x]_i$, $i \in \{1, 2, ..., n\}$, satisfying $\sum_{i=1}^{n}[x]_i = x$, where $x \in \mathbb{R}$. It is an (n, n)-threshold secret sharing scheme as the lack of any share will make the information independent from secret. It naturally supports the linear operations: $\sum_{i=1}^{n}([x]_i \pm [y]_i) = x \pm y$. It implies each party can execute $x \pm y$ locally, and the result is also in the additive form. Similarly, each party can execute multiplication and division with the constant without any interaction.

To support the multiplication on additive shares, ASS can generate *Beaver triple* [42] $\{a, b, c | c = ab\}$ offline, where a and b are randomly picked up from $\mathbb{R}$ such that $c = ab$. According to the following equation:

$$(x - a)(y - b) = xy - a(y - b) - b(x - a) + ab, \quad (1)$$

the information of x and y will be covered by the triple, but the additive shares of xy can be calculated.

The task of generating random numbers during offline phase can be done by a trustworthy party such as the model owner. One possible method for achieving secure multiplication involves utilizing the algorithm presented in [43] which is as follows: Every party P_i computes $[e]_i = [x]_i - [a]_i$ and $[f]_i = [y]_i - [b]_i$ locally. Next, they collaboratively recover e and f. Then, every party P_i computes $[xy]_i = [c]_i + ([b]_i \times e) + ([a]_i \times f)$ locally. Finally, the party P_1 should update the value of $[xy]_1$ using $[xy]_1 = [xy]_1 + (e \times f)$. At this time, every party P_i has the secret share of $[xy]_i$, where $xy = \sum_{i=1}^{n}[xy]_i$.

III. Proposed Framework

In this section, we explain about SafeML framework in details.

A. System Architecture

The system architecture of SafeML is given in Fig. 1. SafeML includes a proxy layer of computing nodes that act as an intermediary between data owners and model owners. When using SafeML for model training, the following actors come into play:

- Data Owners: A set of m data owners $\{DO_1, DO_2, ..., DO_m\}$ who hold multiple samples with a consistent set of features and seek to securely outsource them for model training.
- Model Owner: Model Owner is tasked with developing and maintaining the machine learning model. This can be carried out by individuals, teams, or third-party companies.
- Computing Nodes: They perform model training operations or validate computation results and aggregate them when necessary.
- Adversaries: They act as Byzantine nodes, intentionally providing incorrect computation results or colluding with other Byzantine nodes to gain access to sensitive information, and may also tamper with data.

The proxy layer in SafeML comprises the following computing nodes:

- Worker Nodes: A Set of N groups of worker nodes, with each group G_i comprising P_i worker nodes $\{WN_1, WN_2, ..., WN_{P_i}\}$. In a given group, the worker nodes have the responsibility of carrying out identical computations throughout the model training process.
- Auxiliary Nodes: A set of q auxiliary nodes $\{AN_1, AN_2, ..., AN_q\}$ which assist the worker nodes in performing linear operations such as matrix multiplication that cannot be computed locally. They act as a facilitator and communicate with every individual worker node to accomplish this.
- Mediator Nodes: A set of r mediator nodes $\{MN_1, MN_2, ..., MN_r\}$ that act as aggregators in the proxy layer. They collect and aggregate the correct computation results produced by worker nodes to produce the final output.

To initiate model training, data owners share their input data with every worker nodes in a secret-shared manner, ensuring that every worker node in a group has access to the same set of secret shares without revealing the actual input data. Similarly, the model owner shares the security parameters and

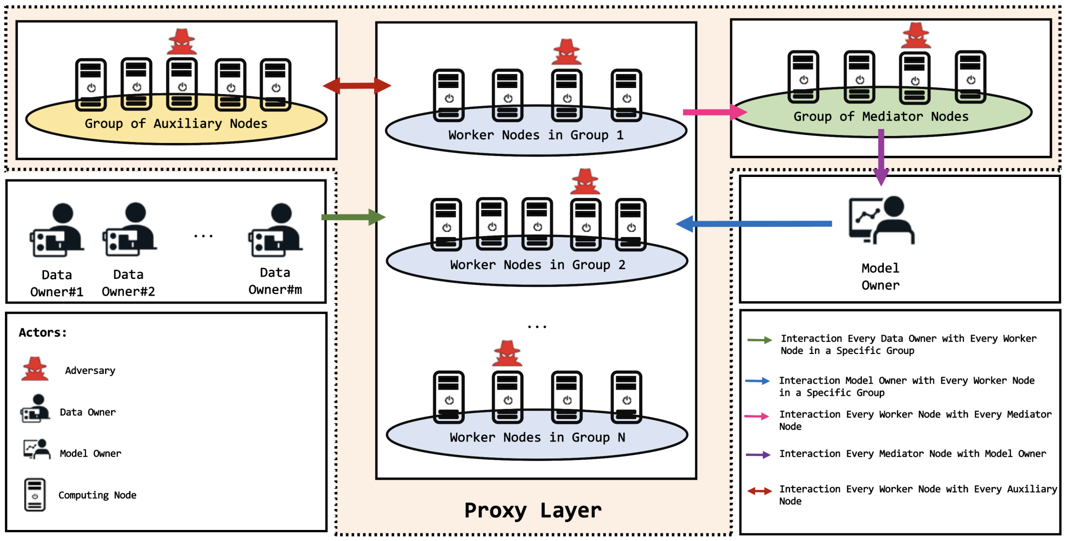

Group of Auxiliary Nodes
Worker Nodes in Group 1
Group of Mediator Nodes
Data Owner#1
Data Owner#2
...
Data Owner#m
Worker Nodes in Group 2
Model Owner
...
Worker Nodes in Group N
Actors:
Adversary
Data Owner
Model Owner
Computing Node
Interaction Every Data Owner with Every Worker Node in a Specific Group
Interaction Model Owner with Every Worker Node in a Specific Group
Interaction Every Worker Node with Every Mediator Node
Interaction Every Mediator Node with Model Owner
Interaction Every Worker Node with Every Auxiliary Node
Proxy Layer

Fig. 1: System Architecture of SafeML

model parameters as secret shares with worker nodes. With the help of auxiliary nodes and a robust confirmation process, every individual worker node can securely and independently perform the majority of model training operations, without requiring assistance from the model owner. The mediator nodes in the proxy layer receive the masked computation result from every individual worker node, and identify the maximum occurrence of a value as the result of computation for every group of worker nodes. They aggregate the results of every group to obtain the correct masked result, which is then sent to the model owner. The design of SafeML enables the model owner to choose how the worker nodes conceal their computation results using the secret shares of security parameters generated by him/her. In SafeML, computational redundancy and robust confirmation are implemented among each group of computing nodes to mitigate the risks posed by potential Byzantine nodes within each group.

B. ASS-based Protocols

To secure all computations and filter out incorrect computations during model training, SafeML uses a set of ASS-based protocols intermixed with a robust confirmation as follows:

- *Secure Addition Protocol* (SecAdd): Given the random shares ($[A]_i$, $[B]_i$) of two inputs $A \in \mathbb{R}$ and $B \in \mathbb{R}$ to worker nodes of the group G_i in the proxy layer, where $1 \leq i \leq N$, every worker node performs the secure addition locally and returns share $[C]_i$ where $[C]_i = [A]_i + [B]_i$ and $\forall 1 \leq i \leq N, [C]_1 + [C]_2 + ... + [C]_N = A + B$.
- *Secure Subtraction Protocol* (SecSub): Given the random shares ($[A]_i$, $[B]_i$) of two inputs $A \in \mathbb{R}$ and $B \in \mathbb{R}$ to worker nodes of the group G_i in the proxy layer, where $1 \leq i \leq N$, every worker node performs the secure subtraction locally and returns share $[C]_i$ where $[C]_i = [A]_i - [B]_i$ and $\forall 1 \leq i \leq N, [C]_1 + [C]_2 + ... + [C]_N = A - B$.
- *Secure Division Protocol* (SecDiv): Given a division factor $k \in \mathbb{R}$ and the random shares $[A]_i \in \mathbb{R}$ to worker nodes of the group G_i in the proxy layer, where $1 \leq i \leq N$, every worker node performs the secure division locally and returns share $[C]_i$ where $[C]_i = [A]_i/k$ and $\forall 1 \leq i \leq N, [C]_1 + [C]_2 + ... + [C]_N = A/k$.
- *Secure Multiplication Protocol* (SecMul): Given the random shares ($[A]_i$, $[B]_i$) of two inputs $A \in \mathbb{R}$ and $B \in \mathbb{R}$ to worker nodes of the group G_i in the proxy layer, where $1 \leq i \leq N$, every worker node performs secure multiplication using the method explained in Section II intermixed with a robust confirmation as follows: Firstly, every worker node of the group G_i computes $[e]_i$ and $[f]_i$ locally and sends them to the group of auxiliary nodes. The auxiliary nodes obtain the most frequently occurring values of $[e]_i$ and $[f]_i$. These values are used to reconstruct e and f, which are sent back to every worker node of the group G_i. Then, every worker node of the group G_i obtains the values of e and f with the highest frequency of occurrence and uses them to compute the share $[C]_i$, where $[C]_i = [AB]_i$, and $\forall 1 \leq i \leq N$, $[C]_1 + [C]_2 + ... + [C]_N = A \times B$.
- *Secure Matrix Multiplication Protocol* (SecMatMul): Given the random shares ($[A]_i$, $[B]_i$) of two input matrices $A \in \mathbb{R}^{u \times p}$ and $B \in \mathbb{R}^{p \times v}$ to worker nodes of the group G_i in the proxy layer, where $1 \leq i \leq N$, every worker node performs secure multiplication using the method explained in Section II intermixed with a robust confirmation, analogous to SecMul protocol (performing matrix operations instead of operations on numbers). It then returns shares $[C]_i$ where $[C]_i = [AB]_i$ and $\forall 1 \leq i \leq N, [C]_1 + [C]_2 + ... + [C]_N = A \times B$.
- *Secure Comparison Protocol* (SecComp): When comparing two numbers, only the sign of the difference between the numbers is necessary. However, the magnitude of the

difference between the numbers does not provide any meaningful information about the numbers themselves. Given the random shares ($[A]_i$, $[B]_i$) of two inputs $A \in \mathbb{R}$ and $B \in \mathbb{R}$ and the random share $[T]_i$ of positive random number $T \in \mathbb{R}^+$ to worker nodes of the group G_i in the proxy layer, where $1 \leq i \leq N$, every worker node in the group G_i first computes $[\alpha]_i = [A]_i - [B]_i$ locally and uses it to compute $[T\alpha]_i$ using *SecMul*$([T_i], [\alpha]_i)$. Then, every worker node in the group G_i sends the value $[T\alpha]_i$ to the group of auxiliary nodes. The auxiliary nodes obtain the most frequently occurring value of $[T\alpha]_i$ and uses it to reconstruct $T\alpha$, which is sent back to every worker node of the group G_i. Finally, every worker node of the group G_i obtains the value of $T\alpha$ with the highest frequency of occurrence and uses it to determine *sign*$(T\alpha)$, where *sign*$(T\alpha) =$ *sign*$(A - B)$.

C. Model Training in SafeML

Most machine learning models require three types of operations during model training, which can be categorized as follows:

- *Linear Operations*: These operations involve basic arithmetic operations such as addition, subtraction, division, multiplication, as well as matrix multiplication.
- *Non-Linear Element-Wise Functions*: During model training, activation functions are used to introduce non-linearity to the output of a linear operation.
- *Local Transformations*: These operations involve modifying the shape of an input data structure to make it compatible with a specific layer or component of a machine learning model.

ASS-based protocols provide an efficient and straightforward means of implementing linear operations. The local transformations can be easily performed by every individual worker node on its secret shares. However, activation functions are typically non-linear and cannot be implemented using ASS-based protocols. SafeML addresses this challenge through two solutions. Firstly, activation functions can be substituted with polynomial approximations in the form of $f(x) = a_0 + a_1x + a_2x^2 + ... + a_dx^d$, which can be efficiently computed by ASS-based protocols and applied to commonly used activation functions such as *Sigmoid* and *Tanh*. While this approach could potentially result in a reduction of model accuracy, it minimizes the involvement required from the data owner throughout the learning process in SafeML. Moreover, the ReLU activation function can be computed using the *SecComp* protocol, while polynomial approximations are available for such an activation function. Secondly, in cases where an approximate polynomial is not available, such as with the *Softmax* function, the model owner can be requested to perform the activation function. Furthermore, differentiation of the activation function ($\frac{da}{dx}$) can be easily computed using either of these solutions.

By employing these operations, SafeML can successfully train a diverse range of machine learning models. As an illustration, Algorithm 1 offers a case study demonstrating how SafeML can be employed to train a Back-Propagation Multilayer Perceptron neural network (BP-MLP) model.

Algorithm 1: MLP Classification Training (SGD)

```
Input: Worker nodes of the group $G_i$ hold additive shares of data $[X]_i$ and
       label $[Y]_i$, additive shares of network parameters (Weights, Bias)
       $\{[W_j]_i, [b_j]_i\}$ for every layer $j$, additive shares of random masks
       $[\widetilde{W}_j]_i$ and $[\widetilde{b}_j]_i$ for every layer $j$, and an additive share $[\widetilde{Y}]_i$ of a
       random output mask $\widetilde{Y}$ from the model owner.
Output: Worker nodes of the group $G_i$ obtain additive shares of the updated
        model parameters $\{[W'_j]_i, [b'_j]_i\}$ for every layer $j$.

   /* Forward Pass                                                    */
 1 $[A_0]_i = Flatten([X]_i)$;
 2 for $j = 0, \dots, (NumLayers - 1)$ do
 3     $[Z_{j+1}]_i = SecAdd(SecMatMul([W_j]_i, [A_j]_i), [b_j]_i)$;
 4     if $j < (NumLayers - 1)$ then
           // ReLU activation in the hidden layers
 5         $[A_{j+1}]_i, sign_{Z_{j+1}} = ReLU([Z_{j+1}]_i)$ ;          // see
            algorithm 2
 6     else
           // Softmax activation in the output layer
 7         $[A_{j+1}]_i = Softmax([Z_{j+1}]_i, [\widetilde{Y}]_i)$ ;  // see algorithm 3

 8 $[\hat{Y}]_i = [A_{NumLayers}]_i$;
   /* Backward Pass                                                   */
   // assume cross-entropy loss
 9 $[e_{NumLayers}]_i = SecSub([\hat{Y}]_i, [Y]_i)$;
10 for $j = (NumLayers - 1), \dots, 0$ do
       // row-wise multiplication $A * b$ with SecMul
11     $[\Delta W_j]_i = [W_j]_i * [e_{j+1}]_i$;
12     $[\Delta b_j]_i = [e_{j+1}]_i$;
       // ReLU back-propagation
13     $[e_{j+1}]_i[sign_{Z_{j+1}} < 0] = 0$;
14     $[e_j]_i = SecMatMul(Transpose([W_j]_i), [e_{j+1}]_i)$;

   /* Model Update                                                    */
15 for $j = 0, \dots, (NumLayers - 1)$ do
16     $[\overline{\Delta W_j}]_i = SecSub([\Delta W_j]_i, [\widetilde{W}_j]_i)$;
17     $[\overline{\Delta b_j}]_i = SecSub([\Delta b_j]_i, [\widetilde{b}_j]_i)$;
18 Send all $[\overline{\Delta W_j}]_i$ and $[\overline{\Delta b_j}]_i$ to the group of mediator nodes;
19 Every mediator node obtains each $[\overline{\Delta W_j}]_i$ and $[\overline{\Delta b_j}]_i$ as the values with the
     highest frequency of occurrence, reconstructs all $\overline{\Delta W_j}$ and $\overline{\Delta b_j}$, and sends
     them to the model owner;
20 The model owner obtains all $\overline{\Delta W_j}$ and $\overline{\Delta b_j}$ as the values with the highest
     frequency of occurrence and computes $\Delta W_j = \overline{\Delta W_j} + \widetilde{W}_j$ and
     $\Delta b_j = \overline{\Delta b_j} + \widetilde{b}_j$;
21 The model owner updates all local parameters as $W'_j = W_j - \eta \Delta W_j$ and
     $b'_j = b_j - \eta \Delta W_j$ with learning rate $\eta$, creates additive shares $[W'_j]_i, [b'_j]_i$
     for the new parameters and sends them to worker nodes of the group $G_i$;
```

Algorithm 2: ReLU Activation Protocol (*ReLU*)

```
Input: Worker nodes of the group $G_i$ hold an additive share $[Z]_i$ of
       activation input $Z$.
Output: Worker nodes of the group $G_i$ obtain an additive share of the ReLU
        activation and the signs ($sign_Z$) of the input $Z$.
  // batch-wise SecComp
1 $sign_Z = SecComp([Z]_i, \mathbf{0})$;
  // $\{sign_Z < 0\}$ denotes all indices where $sign_Z$ is
     negative
2 $[Z]_i\{sign_Z < 0\} = 0$;
3 return $[Z]_i, sign_Z$;
```

IV. Security Analysis

The security of ASS-based protocols can be proven under the *honest-but-curious* adversary model by utilizing the universal composability framework [44]. We can follow the same direction as stated in [45], [46] to prove their security. However, to save space, we assume that the security of these

Algorithm 3: Softmax Activation Protocol (*Softmax*)

```
Input: Worker nodes of the group $G_i$ hold an additive share $[Z]_i$ of
       activation input $Z$ and an additive share $[\widetilde{Y}]_i$ of a random output
       mask $\widetilde{Y}$ from the model owner.
Output: Worker nodes of the group $G_i$ obtain an additive share of the
        Softmax activation of input $Z$.
1 Send $[\overline{Z}]_i = SecSub([Z]_i, [\widetilde{Y}]_i)$ to the group of mediator nodes;
2 Every mediator node obtains each $[\overline{Z}]_i$ as the value with the highest
    frequency of occurrence, reconstructs $\overline{Z}$ and sends it to the model owner;
3 The model owner obtains all $\overline{Z}$ as the value with the highest frequency of
    occurrence, reconstructs $Z = \overline{Z} + \widetilde{Y}$ and computes
    $A\{k\} = \frac{\exp(Z\{k\})}{\sum_{l=1}^{K}\exp(Z\{l\})}$ for each output component $A\{k\}$ of the
    Softmax output $A$ with $K$ classes;
4 The model owner creates additive shares $[A]_i$ for $A$ and sends them to
    worker nodes of the group $G_i$;
5 return $[A]_i$;
```

protocols has been proven and only focus on demonstrating the security of SafeML during model training.

In the *honest-but-curious* adversary setting, we assume that the computing node groups (including worker, auxiliary, or mediator nodes) are hosted across diverse cloud infrastructures. Each group is deployed on a specific cloud infrastructure managed by a distinct cloud service provider. Additionally, we assume that the majority of these cloud service providers refrain from colluding with one another due to their conflicting interests. Nevertheless, in this setting, we make a worst-case assumption that any worker node in $N-1$ groups, as well as all auxiliary and mediator nodes in the proxy layer, could potentially act as the adversary (denoted as $\mathcal{A}$) and may even collude with each other. The security of SafeML in this setting is established by proving that the ideal experiment, where a simulator (denoted as $\mathcal{S}$) emulates $\mathcal{A}$'s view according to the functionality $\mathcal{F}$, is indistinguishable from the real experiment. To achieve this, we assume that none of the worker nodes in group G_N are involved in collusion, as there is at least one honest cloud service provider hosting the worker nodes in group G_N. This assumption is also widely adopted in related literature [7], [8], [43], [45], [46], where only one computing node is assumed to be hosted in a particular cloud infrastructure. The functionality $\mathcal{F}$ is defined such that a trusted functionality machine has access to the correct input information required to complete all operations involved in model training. The ideal experiment aims to fill $\mathcal{S}$'s view with the computation results. We demonstrate that this view is indistinguishable from the real-world view, ensuring that the sensitive information remains concealed.

Theorem 4.1: SafeML is secure in the *honest-but-curious* adversary model.

Proof 4.1: During model training, the view of adversary can be defined as $view = ([ID]_i, [MP_j]_i, [MP'_j]_i, [SP_j]_i, [IR_j]_i, [MR_j]_k)$, where:

- $[ID]_i$ represents the ith secret share of the input data, including $[X]_i$ and $[Y]_i$.
- $[MP_j]_i$ represents the ith secret share of model parameters in the jth layer, including $[W_j]_i$ and $[b_j]_i$.
- $[MP'_j]_i$ represents the ith secret share of updated model parameters in the jth layer, including $[W'_j]_i$ and $[b'_j]_i$.
- $[SP_j]_i$ represents the ith secret share of security parameters, including $[\widetilde{W_j}]_i$ and $[\widetilde{b_j}]_i$ in the jth layer, and $[\widetilde{Y}]_i$.
- $[IR_j]_i$ represents the ith secret share of intermediate results generated by every worker node of the group G_i such as $[Z_{j+1}]_i$, $[A_{j+1}]_i$, $[e_{j+1}]_i$, $[\Delta W_j]_i$ and $[\Delta b_j]_i$.
- $[MR_j]_k$ represents the kth masked result generated by every worker node of the group G_i for the jth layer, including $[\overline{\Delta W_j}]_k$ and $[\overline{\Delta b_j}]_k$.

for $1 \le i \le (N-1)$, $0 \le j \le (NumLayer - 1)$, and $1 \le k \le N$.

To recover ID, MP_j, MP'_j, SP_j, and IR_j, the adversary $\mathcal{A}$ would need access to $[ID]_N$, $[MP_j]_N$, $[MP'_j]_N$, $[SP_j]_N$, and $[IR_j]_N$, respectively. However, these secret shares are not accessible to the adversary $\mathcal{A}$. Moreover, even though the adversary $\mathcal{A}$ can recover MR_j, which includes $\overline{\Delta W_j}$ and $\overline{\Delta b_j}$, they cannot determine the exact values of deterministic results (i.e., ΔW_j and Δb_j) due to the absence of $[SP_j]_N$ (i.e., $[\widetilde{W_j}]_N$ and $[\widetilde{b_j}]_N$). Therefore, the values ID, MP_j, MP'_j, SP_j, IR_j, and MR_j are completely random and simulatable. Consequently, both the view and the output of the protocol are simulatable by the simulator $\mathcal{S}$, and the views of $\mathcal{S}$ and $\mathcal{A}$ are computationally indistinguishable.□

In the *malicious* adversary setting, we maintain the prior assumption that the computing node groups within the SafeML's proxy layer are hosted across diverse cloud infrastructures. Each group is deployed on a distinct cloud infrastructure, managed by a separate cloud service provider, under the premise of honest behavior to safeguard their reputations. However, we postulate that a minority of computing nodes within specific groups, hosted on particular cloud infrastructure, may deviate from honesty. These nodes, known as Byzantine nodes, deliberately supply incorrect computation results and manipulate data. It is noteworthy that this minority of nodes is under the control of the adversary, aiming to compromise computation integrity or tamper with data. The security of SafeML within this *malicious* adversary model is established by demonstrating that every honest computing node in the proxy layer as well as the model owner, can still derive correct computation results from the received computation results. In cases where a computing node within a specific group or the model owner receives erroneous results, they can effectively reject these by identifying the most frequently occurring value as the correct one.

Theorem 4.2: SafeML is secure in the *malicious* adversary model.

Proof 4.2: Let $\mathcal{M}_i$ and $\mathcal{M}_j$ represent the sets of malicious computing nodes, and $\mathcal{H}_i$ and $\mathcal{H}_j$ represent the sets of honest computing nodes in the groups G_i and G_j within the SafeML's proxy layer. Here, G_i and G_j can refer to the groups of worker nodes, auxiliary nodes, or mediator nodes. In the *malicious* adversary model, our assumption is that the majority of computing nodes in the groups G_i and G_j are honest ($|\mathcal{H}_i| > |\mathcal{M}_i|$ and $|\mathcal{H}_j| > |\mathcal{M}_j|$). This assumption holds because these two groups are hosted across different cloud infrastructures that exhibit honest behavior due to their reputation. However, we also assume that the adversary $\mathcal{A}$ can gain control over the minority of computing nodes in the groups G_i and G_j.

Consider the set C_i, which represents computation results received by the group G_i within the SafeML framework. This group includes the group of auxiliary nodes assisting the worker nodes of a specific group to perform a linear operation, the worker nodes of a specific group receiving the computation results from the the group of auxiliary nodes, the group of mediator nodes aggregating the computation results from the worker nodes of different groups and sending the masked computation results to the model owner. The group G_i encompasses both honest and malicious computing nodes.

Honest computing nodes in $\mathcal{H}_i$ consistently compute and share correct results with the computing nodes in the group G_j, while malicious computing nodes in $\mathcal{M}_i$ might introduce incorrect results and send them to the computing nodes in the group G_j. Due to the majority of honest computing nodes in the group G_i, the most frequent value in the set C_i corresponds to the correct computation results. Honest computing nodes in $\mathcal{H}_j$ can identify the correct result using a majority voting mechanism. This ensures that even if malicious computing nodes attempt to manipulate the computation, their influence is limited and can be mitigated through this voting process.

The presence of malicious computing nodes introduces potential errors, but the collective behavior of honest computing nodes in the groups G_i and G_j in the SafeML's proxy layer ensures the overall integrity of the computation process. Honest computing nodes in $\mathcal{H}_j$ can recognize and discard malicious influence due to their majority, enabling the correct results to prevail.$\square$

The presence of honest behavior among cloud service providers, driven by their concern for maintaining their reputation, in conjunction with the cooperation of the majority of honest nodes and the implementation of a majority voting mechanism, collectively ensures the security of SafeML against malicious adversaries. Although adversaries can infiltrate the system and return incorrect computation results or tamper with the data, the collective behavior of the system guarantees both the correctness and security of SafeML for model training in the *malicious* adversary model.

It is important to note that SafeML is not resistant to the assumption that most computing nodes can exhibit dishonest behavior. However, this assumption does not apply to solutions utilizing secret sharing schemes for privacy-preserving machine learning [6]–[9], as these solutions depend on the presumption that most computing nodes adhere to truthfulness.

Remark 4.1: If SafeML can effectively defend against Byzantine nodes, then the model updates after each iteration will be equivalent to those obtained in the Byzantine-resilient setup. This implies that any guarantees of convergence in the Byzantine-resilient setup can be directly applied to the Byzantine case.

V. Performance Evaluation

In this section, we present the preliminary experimental results to demonstrate the applicability of SafeML to train a BP-MLP model.

TABLE I: Datasets and Parameters

Parameters	MNIST	SVHN
Examples per Epoch	60,000	73,257
Epochs	5	5
Architecture	[784]-256-128-[10]	[3072]-256-128-[10]
Learning Rate	0.01	0.002

A. Experiment Setup

We implemented SafeML on Ubuntu 20.04 operating system using the Pytorch and Ray frameworks. All the experiments were performed on a computer with an Intel Core i7-7700 CPU, an NVIDIA GeForce GTX 1060 6 GB GPU, and 16 GB RAM.

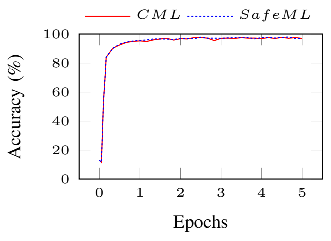
We used two widely-used datasets, MNIST [47] and SVHN [48], in our experiments. The MNIST dataset consists of 70,000 handwritten digit images of size 28 $\times$ 28, with 60,000 images used for training and the remaining 10,000 for testing. The SVHN dataset contains 73,257 color images of size 32 $\times$ 32. Each image is centered around a digit, which represents a house number captured from Google's street view images.

Table I shows the training parameters including the number of examples per epoch (size of the training set), number of epochs (number of training rounds), architecture (given as [number of input neurons], hidden layer sizes, [number of output neurons]; e.g. [784]-256-128-[10] for 784 input neurons, two hidden layers with 256 and 128 neurons respectively, and 10 output neurons) and learning rate used in the BP-MLP models for both datasets. All models were trained using stochastic gradient descend. The weights were initialized randomly using a normal distribution $\mathcal{N}(0, 1/S_W)$, where S_W is the sum of the dimensions of a weight matrix W. In each experiment, we used the same initial weights for the Centralized plaintext Model Learning (CML) approach and SafeML. Feature values in each dataset were normalized between [0, 1]. Additionally, all experiments on SafeML were conducted with N = 2, splitting each secret into two additive shares.

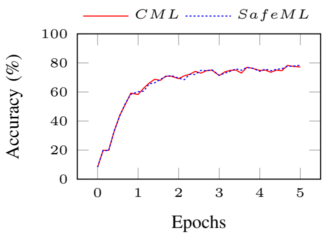
B. Experimental Results

1) Model Accuracy: The experiments aimed to assess the impact of SafeML's privacy protection measures on accuracy compared to CML. Fig. 2 illustrates the accuracy of the trained model in SafeML and CML, and the curves are almost indistinguishable for both datasets, implying that SafeML does not incur significant accuracy loss. This can be attributed to the usage of the ReLU activation function instead of its approximate polynomial and the utilization of 64-bit fixed point integers and 32 precision bits to minimize the accuracy loss resulting from the conversion between float-point and fixed point during model training.

2) Computation Cost: The experiments aimed to assess the impact of SafeML's privacy protection measures on computation time compared to CML. Table II shows the average computation time per epoch in both frameworks. SafeML,



(a) MNIST Dataset



(b) SVHN Dataset

Fig. 2: Accuracy in BP-MLP Model Training

TABLE II: Average Computation Time for Model Training

Framework	MNIST	SVHN
CML	155.883 s	458.008 s
SafeML	714.307 s	2735.778 s

using one node per group, required about 4.6 and 6 times more computation time than CML for model training in the MNIST and SVHN datasets, respectively. Despite the longer computational time in SafeML, the research literature has shown that using secret sharing schemes for privacy-preserving machine learning has a lower computational time than other cryptographic approaches [49], [50]. In general, the benefits of protecting sensitive data during model training outweigh the costs associated with the increased computation time.

3) Communication Cost: The experiments aimed to assess the impact of SafeML's Byzantine robustness on network traffic. Table III presents the network traffic per epoch for model training in SafeML. It is clear that the network traffic in the 3NpG (three computing nodes per group) configuration is approximately 8.5 times higher than in the 1NpG (one computing node per group) configuration for both datasets. This demonstrates that the Byzantine robustness measures in SafeML significantly increase the communication cost. Consequently, it is imperative to devise targeted solutions to reduce the communication cost associated with SafeML.

TABLE III: Network Traffic for Model Training

Config	MNIST	SVHN
1NpG	4427.34 GB	18838.67 GB
3NpG	37814.40 GB	160888.51 GB

VI. Related Works

In this section, we first provide an overview of the current solutions used for the development of distributed machine learning frameworks with a focus on privacy protection and verifiability. Then, we compare them with the techniques used in SafeML.

A. Privacy-Preserving Machine Learning

Privacy-preserving machine learning methods can be broadly categorized into two classes: provable secure methods and non-provable secure methods. Provable secure methods, which rely on cryptographic primitives like Homomorphic Encryption [51]–[54] and Multiparty Computation [55]–[58], ensure that the adversary cannot obtain any information about the input data within a certain time frame, even under different adversary models. In contrast, non-provable secure methods such as federated learning [59], [60] and split learning [61]–[63] may reveal some information about the input data, and they typically lack a formal proof of security. While provable secure methods offer better privacy guarantees, they often suffer from high computational and communication costs, making them challenging to implement in practice. On the other hand, non-provable secure methods are generally more efficient but may still leak privacy under certain conditions. Furthermore, it can be challenging to quantify the extent of the privacy leakage in these methods.

B. Verifiable Machine Learning

Verifiable computing (VC) is a powerful cryptographic building block that guarantees the integrity or correctness of computations performed by a server. VC allows a client to confirm that the server has correctly performed the computations by providing not only the computation results but also a mathematical proof of correctness. Solutions focusing on verifiable machine learning, such as Interactive Proof Systems [17], [29], Homomorphic Encryption [30], [31], and Zero-Knowledge Proofs [32]–[34] have been proposed, but they suffer from high computational and communication overhead and complexity.

Another robust solution to ensure the privacy of data and integrity of computation during model training involves leveraging a Trusted Execution Environment (TEE). TEE-based solutions [13], [35], [64], [65] necessitate the client to place trust in the hardware manufacturer and assume the hardware's tamper-proof nature. In these solutions, during the training phase, sensitive data is processed within the TEE, providing a shield against unauthorized access and tampering. This fortified environment contributes to the safeguarding of data privacy and the assurance of both data integrity and the integrity of computation results. However, it is important to acknowledge that TEE-based solutions, while effective, are not impervious and remain potentially susceptible to challenges such as malware attacks [66] and side-channel attacks [36]–[41].

C. Comparison with Existing Solutions

The SafeML framework proposed in this paper employs secret sharing and data masking techniques to ensure the privacy of input data, model parameters, and intermediate results generated during the training process. Compared to non-provable secure methods like federated learning and split learning, as well as TEE-based solutions, which reveal certain information about input data and computation results during model training [9], SafeML offers stronger privacy guarantees, as proven through our security analysis in Section IV. Furthermore, similar to other secret-sharing-based solutions [7], [8], SafeML conducts all operations on secret-shared data using simple mathematical operations such as addition, subtraction, division, and multiplication. Therefore, SafeML exhibits greater efficiency and lower computational costs than cryptographic-based methods, including Homomorphic Encryption and Multiparty Computation, in the context of model training.

To detect malicious behavior and ensure robustness against Byzantine attacks during model training, SafeML employs computational redundancy and robust confirmation methods as recovery measures. These measures help SafeML to detect malicious behavior of computing nodes and filter out incorrect intermediate results returned by Byzantine nodes. SafeML determines the most frequently occurring value as the correct value during model training. This computation requires less processing compared to verifiable computing methods like Interactive Proof Systems, Homomorphic Encryption, and Zero-Knowledge Proofs, which are used to reach an agreement on computation results. However, similar to all secret-sharing-based solutions [6]–[9], SafeML suffers from high communication costs due to the need for interaction between different computing nodes during model training.

Overall, SafeML aims to address the limitations of existing privacy-preserving Byzantine-robust distributed machine learning frameworks and provide a trustworthy solution for real-world machine learning applications. Nevertheless, there exists an avenue for improvement within SafeML – optimizing it to mitigate the communication cost entailed in the process of model training.

VII. Conclusion and Future Works

In this paper, we proposed SafeML, a trustworthy framework that not only preserves the privacy of input data, model parameters, and intermediate results during model training, but also incorporates robustness against Byzantine attacks. Our security analysis confirmed that SafeML is effective in defending against both *honest-but-curious* and *malicious* adversaries. Furthermore, experimental results showed that SafeML can train a BP-CNN model with satisfactory accuracy, while acknowledging the challenge of high communication cost.

Moving forward, we plan to extend our work in several directions, including:

1) Designing a protocol that optimizes SafeML's communication cost by collecting computation results from a subset of computing nodes based on a threshold. This threshold can be set to the maximum number of Byzantine nodes that every group of computing nodes can tolerate.
2) Expanding SafeML to offer privacy-preserving Byzantine-robust inference service.
3) Extending SafeML to support datasets partitioned in arbitrary ways, instead of multiple samples with a consistent set of features.
4) Theoretically analyzing the costs of computation and communication in SafeML for both model training and inference.
5) Conducting comprehensive experiments to evaluate the performance of SafeML for both model training and inference in terms of computation cost, communication cost, and accuracy loss compared to existing solutions.

By addressing these issues, we aim to improve the efficiency of SafeML and expand its scope, making it a versatile and effective distributed machine learning framework.

Acknowledgment

This work was partially funded by the BMWK project SafeFBDC (01MK21002K) and the National Research Center ATHENE. We also want to thank hessian.AI at TU Darmstadt as well as DFKI Darmstadt for the support.

References

[1] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*. Boston, MA, USA: IEEE, 2015, pp. 815–823.

[2] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “Deepface: Closing the gap to human-level performance in face verification,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*. Columbus, OH, USA: IEEE, 2014, pp. 1701–1708.

[3] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, “Dermatologist-level classification of skin cancer with deep neural networks,” *Nature*, vol. 542, no. 7639, pp. 115–118, Feb. 2017.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[5] W. Ouyang, X. Wang, X. Zeng, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, C.-C. Loy, and X. Tang, “Deepid-net: Deformable deep convolutional neural networks for object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*. Boston, MA, USA: IEEE, 2015, pp. 2403–24 128.

[6] Z. Zhou, Q. Fu, Q. Wei, and Q. Li, “Lego: A hybrid toolkit for efficient 2pc-based privacy-preserving machine learning,” *Computers & Security*, vol. 120, p. 102782, Jun. 2022.

[7] J. Duan, J. Zhou, and Y. Li, “Privacy-preserving distributed deep learning based on secret sharing,” *Information Sciences*, vol. 527, pp. 108–127, Apr. 2020.

[8] J. Duan, J. Zhou, Y. Li, and C. Huang, “Privacy-preserving and verifiable deep learning inference based on secret sharing,” *Neurocomputing*, vol. 483, pp. 221–234, 2022.

[9] F. Zheng, C. Chen, X. Zheng, and M. Zhu, “Towards secure and practical machine learning via secret sharing and random permutation,” *Knowledge-Based Systems*, vol. 245, p. 108609, 2022.

[10] Q. Jia, L. Guo, Z. Jin, and Y. Fang, “Preserving model privacy for machine learning in distributed systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 8, pp. 1808–1822, 2018.

[11] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, and B. Feng, “Veriml: Enabling integrity assurances and fair payments for machine learning as a service,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 10, pp. 2524–2540, 2021.

[12] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, “A survey on distributed machine learning,” *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–33, Mar. 2020.

[13] F. Tramer and D. Boneh, “Slalom: Fast, verifiable and private execution of neural networks in trusted hardware,” in *ICLR 2019*, New Orleans, LA, United States, 2019, pp. 10 320–10 330.

[14] Q. Xia, Z. Tao, and Q. Li, “Defenses against byzantine attacks in distributed deep neural networks,” *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2025–2035, Jul. 2021.

[15] S. Rajput, H. Wang, Z. Charles, and D. Papailiopoulos, “Detox: a redundancy-based framework for faster and more robust gradient aggregation,” in *NeurIPS 2019*, Vancouver, BC, Canada, 2019, pp. 10 320–10 330.

[16] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, “Draco: Byzantine-resilient distributed training via redundant gradients,” in *Proceedings of the 35th International Conference on Machine Learning*. Stockholmsmässan, Stockholm, Sweden: PMLR., 2018, pp. 10 320–10 330.

[17] Z. Ghodsi, T. Gu, and S. Garg, "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud," in *NIPS 2017*, Long Beach, CA, USA, 2017, p. 4675–4684.

[18] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: byzantine tolerant gradient descent," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Long Beach, CA, USA: Curran Associates Inc., 2017, pp. 118–128.

[19] B. Liu, Y. Jiang, F. Sha, and R. Govindan, "Cloud-enabled privacy-preserving collaborative learning for mobile sensing," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*. Toronto, ON, Canada: ACM, 2012, pp. 57–70.

[20] O. L. Mangasarian, "Privacy-preserving linear programming," *Optimization Letters*, vol. 5, pp. 165–172, Feb. 2011.

[21] J. Vaidya, H. Yu, and X. Jiang, "Privacy-preserving svm classification," *Knowledge and Information Systems*, vol. 14, pp. 161–178, Mar. 2008.

[22] 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, no. 5, pp. 1333–1345, May 2018.

[23] L. T. Phong and T. T. Phuong, "Privacy-preserving deep learning via weight transmission," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 11, pp. 3003–3015, Nov. 2019.

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

[25] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. Denver, CO, USA: ACM, 2015, pp. 1322–1333.

[26] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *USENIX security symposium*. Austin, TX, USA: ACM, 2016, pp. 601–618.

[27] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. Abu Dhabi, United Arab Emirates: ACM, 2017, pp. 506–519.

[28] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton, "A methodology for formalizing model-inversion attacks," in *Proceedings of the 2016 IEEE 29th Computer Security Foundations Symposium*. Lisbon, Portugal: IEEE, 2016, pp. 355–370.

[29] S. Goldwasser, G. N. Rothblum, J. Shafer, and A. Yehudayoff, "Interactive proofs for verifying machine learning," in *ITCS 2021*, Virtual Event, 2021, pp. 41:1–41:19.

[30] C. Niu, F. Wu, S. Tang, S. Ma, and G. Chen, "Toward verifiable and privacy preserving machine learning prediction," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 1703–1721, 2020.

[31] A. Hassan, R. Hamza, H. Yan, and P. Li, "An efficient outsourced privacy preserving machine learning scheme with public verifiability," *IEEE Access*, vol. 7, pp. 146 322–146 330, 2019.

[32] Y. Fan, B. Xu, L. Zhang, J. Song, A. Zomaya, and K.-C. Li, "Validating the integrity of convolutional neural network predictions based on zero-knowledge proof," *Information Sciences*, vol. 625, pp. 125–140, 2023.

[33] T. Liu, X. Xie, and Y. Zhang, "Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *CCS '21*, Virtual Event, Republic of Korea, 2021, pp. 2968–2985.

[34] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning," in *Proceedings of the 30th USENIX Security Symposium*. Virtual Event: USENIX Association, 2021, pp. 501–518.

[35] Y. Chen, F. Luo, T. Li, T. Xiang, Z. Liu, and J. Li, "A training-integrity privacy-preserving federated learning scheme with trusted execution environment," *Information Sciences*, vol. 522, pp. 69–79, 2020.

[36] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th European Workshop on Systems Security*. Belgrade, Serbia: ACM, 2017, pp. 1–6.

[37] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "Cachezoom: How sgx amplifies the power of cache attacks," in *Proceedings of International Conference on Cryptographic Hardware and Embedded Systems*. Taipei, Taiwan: Springer, 2017, pp. 69–90.

[38] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: abusing intel sgx to conceal cache attacks," *Cybersecurity*, vol. 3, pp. 1–20, Jan. 2020.

[39] S. Hosseinzadeh, H. Liljestrand, V. Leppänen, and A. Paverd, "Mitigating branch-shadowing attacks on intel sgx using control flow randomization," in *Proceedings of the 3rd Workshop on System Software for Trusted Execution*. Toronto, ON, Canada: ACM, 2018, pp. 42–47.

[40] S. Fei, Z. Yan, W. Ding, and H. Xie, "Security vulnerabilities of sgx and countermeasures: A survey," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–36, 2021.

[41] K. Murdock, D. Oswald, F. D. Garcia, J. V. Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," in *Proceedings of the 2020 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE, 2020, pp. 1466–1482.

[42] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO'91*, Santa Barbara, CA, USA, 1991, pp. 420–432.

[43] Z. Xia, Q. Gu, W. Zhou, L. Xiong, J. Weng, and N. Xiong, "Str: Secure computation on additive shares using the share-transform-reveal strategy," *IEEE Transactions on Computers*, 2021. [Online]. Available: doi: 10.1109/TC.2021.3073171

[44] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS 2001*, Las Vegas, NV, USA, 2001, pp. 136–145.

[45] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," in *PETS 2019*, Stockholm, Sweden, 2019, p. 26–49.

[46] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Vienna, Austria: ACM, 2016, pp. 805–817.

[47] F. Lauer, C. Y. Suen, and G. Bloch, "A trainable feature extractor for handwritten digit recognition," *Pattern Recognition*, vol. 40, no. 6, pp. 1816–1824, Jun. 2007.

[48] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop 2011*, 2011, pp. 1–9.

[49] V. K. Marimuthu and C. Lakshmi, "Performance analysis of privacy preserving distributed data mining based on cryptographic techniques," in *ICEES 2021*, 2021, pp. 635–640.

[50] A. Boulemtafes, A. Derhab, and Y. Challal, "A review of privacy-preserving techniques for deep learning," *Neurocomputing*, vol. 384, pp. 21–45, 2020.

[51] 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 *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. New York, NY, USA: PMLR, 2016, pp. 201–210.

[52] L. Yuan and G. Shen, "A training scheme of deep neural networks on encrypted data," in *Proceedings of the 2020 International Conference on Cyberspace Innovation of Advanced Technologies*. Guangzhou, China: ACM, 2020, pp. 490–495.

[53] Q. Zhang, L. T. Yang, and Z. Chen, "Privacy preserving deep computation model on cloud for big data feature learning," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1351–1362, May 2016.

[54] F. Bu, Y. Ma, Z. Chen, and H. Xu, "Privacy preserving back-propagation based on bgv on cloud," in *Proceedings of IEEE 17th International Conference on High Performance Computing and Communications*. New York, NY, USA: IEEE, 2015, pp. 1791–1795.

[55] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation," in *Proceedings of the 22nd Annual Network and Distributed System Security Symposium*. San Diego, CA, USA: The Internet Society, 2015, pp. 497–512.

[56] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Proceedings of IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE, 2017, pp. 19–38.

[57] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas, TX, USA: ACM, 2017, p. 619–631.

[58] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on*

Asia conference on computer and communications security. Incheon, Republic of Korea: ACM, 2018, pp. 707–721.

[59] X. Yin, Y. Zhu, and J. Hu, "A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–36, Jul. 2021.

[60] C. Jiang, C. Xu, and Y. Zhang, "Pflm: Privacy-preserving federated learning with membership proof," *Information Sciences*, vol. 576, pp. 288–311, Oct. 2021.

[61] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, Aug. 2018.

[62] J. Jeon and J. Kim, "Privacy-sensitive parallel split learning," in *Proceedings of 2020 International Conference on Information Networking*. Barcelona, Spain: IEEE, 2020, pp. 7–10.

[63] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Camtepe, Y. Gao, H. Kim, and S. Nepal, "Can we use split learning on 1d cnn models for privacy preserving training?" in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. Taipei, Taiwan: ACM, 2020, p. 305–318.

[64] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," in *arXiv preprint*, 2018.

[65] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, F. Xu, C. Xu, L. Zhang, and J. Song, "Occlumency: Privacy-preserving remote deep-learning inference using sgx," in *Proceedings of the 25th Annual International Conference on Mobile Computing and Networking (MobiCom '19)*. Los Cabos Mexico: ACM, 2019, pp. 1–17.

[66] M. Schwarz, S. Weiser, and D. Gruss, "Practical enclave malware with intel sgx," in *Proceedings of the 16th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*. Gothenburg, Sweden: Springer, 2019, p. 177–196.