Secure Shapley Value for Cross-Silo Federated Learning

Shuyuan Zheng
Kyoto University
caryzheng@db.soc.i.kyoto-u.ac.jp

Yang Cao
Hokkaido University
yang@ist.hokudai.ac.jp

Masatoshi Yoshikawa
Kyoto University
yoshikawa@i.kyoto-u.ac.jp

ABSTRACT

The Shapley value (SV) is a fair and principled metric for contribution evaluation in cross-silo federated learning (cross-silo FL), wherein organizations, i.e., clients, collaboratively train prediction models with the coordination of a parameter server. However, existing SV calculation methods for FL assume that the server can access the raw FL models and public test data. This may not be a valid assumption in practice considering the emerging privacy attacks on FL models and the fact that test data might be clients' private assets. Hence, we investigate the problem of *secure SV calculation* for cross-silo FL. We first propose *HESV*, a one-server solution based solely on homomorphic encryption (HE) for privacy protection, which has limitations in efficiency. To overcome these limitations, we propose *SecSV*, an efficient two-server protocol with the following novel features. First, SecSV utilizes a hybrid privacy protection scheme to avoid ciphertext–ciphertext multiplications between test data and models, which are extremely expensive under HE. Second, an efficient secure matrix multiplication method is proposed for SecSV. Third, SecSV strategically identifies and skips some test samples without significantly affecting the evaluation accuracy. Our experiments demonstrate that SecSV is 7.2-36.6× as fast as HESV, with a limited loss in the accuracy of calculated SVs.

PVLDB Reference Format:
Shuyuan Zheng, Yang Cao, and Masatoshi Yoshikawa. Secure Shapley Value for Cross-Silo Federated Learning. PVLDB, 16(7): 1657 - 1670, 2023.
doi:10.14778/3587136.3587141

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/teijyogen/SecSV.

1 INTRODUCTION

Personal data are perceived as the new oil of the data intelligence era. Organizations (e.g., banks and hospitals) can use machine learning (ML) on personal data to acquire valuable knowledge and intelligence to facilitate improved predictions and decisions. However, acquiring sufficient personal data for ML-based data analytics is often difficult due to numerous practical reasons, such as the limited user scale and diversity; organizations face considerable risk of privacy breaches by sharing user data. Consequently, large personal data are stored as *data silos* with few opportunities to extract the valuable information contained therein.

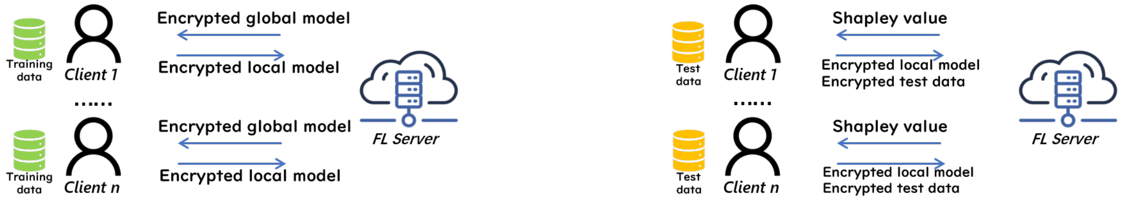
To exploit the data silos in a privacy-preserving manner, *cross-silo federated learning* (cross-silo FL, may be referred to as *cross-organization FL*) [30, 33, 79] was introduced as a promising paradigm for collaborative ML. It enables organizations, i.e., clients, to train an ML model without sharing user data, thus largely protecting privacy. Concretely, in a typical model-training process of FL, each client trains a *local model* on her local side and uploads it to a server. The server then aggregates all the local models into a *global model*, which contains knowledge learned from clients' data silos. However, recent studies have shown that sharing the local models or local updates may reveal private information [11, 80, 86, 88]. Thus, some *secure federated training* systems [25, 41, 57, 60, 73, 82, 84] have deployed *homomorphic encryption* (HE) to prevent the server from accessing the raw models. As shown in Figure 1a, the clients encrypt local models using HE, and the server aggregates the encrypted local models to obtain an encrypted global model that can only be decrypted by the clients.

In typical cases of cross-silo FL, a small number of organizations (e.g., banks and hospitals) collaboratively train an ML model for their own use. Their data may substantially vary in size, quality, and distribution, making their contributions to the model disparate. Therefore, compensations are required to incentivize clients with high contributions to cooperate. In such cases, the Shapley value (SV) [62] is crucial to promoting fair cooperation, which is widely adopted as a fair and principled metric of contribution evaluation. The SV calculates the average impact of a client's data on every possible subset of other clients' data as her contribution and can be used for many downstream tasks in FL or collaborative ML, such as data valuation [13, 22, 23, 34, 76, 77], revenue allocation [17, 36, 53, 69], reward discrimination [65, 70], and client selection [51]. However, existing studies on SV calculation for FL [38, 69, 76, 77] assume that the server can access the raw local models and public test data. This may not be a valid assumption given the emerging privacy attacks on local models [11, 80, 86, 88] and that in practice, test data may be private [9, 21, 53, 56, 75].

Example 1.1. Consider that some hospitals with different geographical distributions of patients collaboratively train an ML model for disease diagnosis by FL. Each hospital provides a training set of its patient data for federated training and a test set for SV-based contribution evaluation. As shown in Figures 1a and 1b, they should ensure the security of both the training and SV calculation phases because the patient data are sensitive.

This study is the first to address *secure Shapley value calculation* in cross-silo FL. Specifically, as depicted in Figure 1b, extending the secure federated training for cross-silo FL [25, 41, 57, 60, 73, 82, 84], we calculate SVs during the evaluation phase using homomorphically encrypted models. Solving this problem is significantly challenging because of the following two characteristics. First, secure SV calculation in cross-silo FL involves protecting both local models

Proceedings of the VLDB Endowment, Vol. 16, No. 7 ISSN 2150-8097.
doi:10.14778/3587136.3587141



(a) Secure federated training.

(b) Secure SV calculation.

Figure 1: Secure federated training and SV calculation for cross-silo federated learning.

and client-side test data. Although the existing studies on SV calculation assume that a public test set is provided, in many practical cases (e.g., [9, 21, 53, 56, 75]), the test data are owned by clients as private assets (see Section 3 for more details). Consequently, we protect both models and test data in our problem, which cannot be supported by existing secure federated training methods because they only protect the model aggregation process. Second, SV calculation is an NP-hard task; it is computationally prohibitive to calculate SVs securely. To calculate SVs in FL, we must aggregate each subset of local models into an *aggregated model* and evaluate its accuracy. This causes $O(2^n)$ models to be tested, where n is the number of clients. Although n is relatively small in the cross-silo setting, secure SV calculation is highly inefficient because even testing only a single model is computationally expensive under HE. Moreover, the existing methods for accelerating SV calculation [10, 13, 23, 43] can hardly improve the efficiency in our setting (as we will show in our experiments) because their strategies are based on sampling and testing a subset of the models to approximate the exact SVs, which is more suitable for large values of n.

1.1 Contributions

As a first step, we propose a one-server protocol for secure SV calculation, named *HESV*. HESV only requires one server to calculate SVs and deploys a purely HE-based scheme for secure model testing, which suggests that both models and test data are encrypted using HE. However, the state-of-the-art (SOTA) homomorphic matrix multiplication method [24] cannot multiply an encrypted matrix of model parameters by an encrypted matrix of input features of a batch of test samples when their sizes are large, which makes evaluating high-dimensional ML models infeasible. Hence, we propose an extended version of the SOTA, named *Matrix Squaring*, which facilitates testing a wider range of models under HESV. Nevertheless, HESV has considerable limitations in efficiency: it involves computationally expensive *ciphertext–ciphertext* (c2c) multiplications between encrypted models and data; it cannot encrypt together test samples from different clients in a single ciphertext to accelerate matrix multiplications; it evaluates the entire test set for all aggregated models, which is time-consuming.

Subsequently, we propose *SecSV*, a two-server protocol that overcomes the limitations of HESV. SecSV is considerably more efficient than HESV, despite requiring an additional server to assist in secure SV calculation. This can be attributed to the following features. First, we design a hybrid protection scheme for SecSV: models are encrypted using HE, whereas test data are protected by *additive secret sharing* (ASS) [61]. Since test data protected by ASS are in plaintext, SecSV only calls for *ciphertext–plaintext* (c2p) multiplications for model testing, which are significantly more efficient than c2c multiplications. Second, owing to the use of ASS, this hybrid scheme enables packing together and simultaneously evaluating numerous test samples from different clients to enhance efficiency. Leveraging this feature, we propose a matrix multiplication method, *Matrix Reducing*, which is significantly more efficient than Matrix Squaring when numerous test samples are batched together (i.e., the case under SecSV). Third, we propose an acceleration method for secure SV calculation called *SampleSkip*. Our intuition is simple yet powerful: if some test samples can be correctly predicted by two models, these samples are likely to be correctly predicted by their aggregated model and thus skippable; otherwise, these samples discriminate the models' utilities and clients' contributions. As test data are stored as plaintext under ASS, we can freely drop skippable samples from the batch for the aggregated model to considerably improve efficiency. Whereas existing SV acceleration methods [10, 13, 23, 43] can hardly reduce the scale of model evaluations in cross-silo FL, SampleSkip always skips massive samples for testing and can be combined with these methods.

Finally, we extensively verify the efficiency and effectiveness of our proposed techniques on diverse ML tasks such as *image recognition*, *news classification*, *bank marketing*, and *miRNA targeting*. SecSV achieves 7.2-36.6× speedup w.r.t. HESV in our experiments.

2 PRELIMINARY

Machine learning task. In this paper, we focus on *classification*, which is an ML task commonly considered in FL and covers a wide range of real-world applications. A c-class classifier is a prediction function $f : R^d \rightarrow R^c$ that given a d-sized *feature vector* x of a data sample yields a c-sized *score vector* $f(x)$; the argument $\hat{y} = \arg\max_j f(x)[j]$ is assigned as the predicted label for features x, where $f(x)[j]$ denotes the j-th entry of $f(x)$. When a batch of m samples is given, we overload the prediction function f for classifying them: for a $d \times m$ *feature matrix* X, where the k-th column is the feature vector of the k-th sample, the output $f(X)$ is a $c \times m$ *score matrix* where the k-th column corresponds to the score vector for the k-th sample, and the predicted labels are elements of an m-sized vector $\hat{Y} = \mathbf{Argmax}(f(X))$, where **Argmax** returns the indices of the maximum values along columns of a given matrix.

A learning task of classification is to seek a classifier f_θ in a *function space* $F = \{f_\theta | \theta \in \Theta\}$, where θ is a set of parameters in a space Θ. We refer to θ as *model parameters* or simply *model*. The task essentially is to find a model θ that optimizes some objective.

Federated learning. FL [44] is a framework for collaborative ML consisting of n clients and an FL server. Concretely, in each training round t, the server selects a subset I^t of clients and broadcasts a *global model* θ^t among selected clients. Each client $i \in I^t$ then trains θ^t using its own training data to derive a *local model* θ_i^t and uploads it to the server. The server aggregates all the local models $\Theta_{I^t} = \{\theta_i^t \mid i \in I^t\}$ into a new global model $\theta^{t+1} = \sum_{i \in I^t} \omega_i^t \theta_i^t$, where $\omega_i^t \geq 0$ denotes the aggregation weight assigned to client i for round t. Finally, after finishing T training rounds, all clients obtain a final model θ^{T+1}.

Shapley value. The SV [62] is a classic metric for evaluating a player's contribution to a coalition in collaborative game theory. It satisfies certain plausible properties in terms of fairness, including balance, symmetry, additivity, and zero element. Given clients $I = \{1, ..., n\}$, the SV measures the expected marginal utility improvement by each client i over all subsets S of I:

$$SV_i = \sum\nolimits_{S \subseteq I \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} \big(v(S \cup \{i\}) - v(S)\big),$$

where $v(\cdot)$ is some utility function of a set of clients.

Neural network. We provide an abstraction of *neural network* (NN), the type of classifier considered throughout this paper. Consider a batch of samples with a feature matrix X. An NN f_θ consists of L layers, where the l-th layer is a linear function $lin^{(l)}$ of the model parameters $\theta^{(l)} \subseteq \theta$ and input features $X^{(l)}$ of the layer, where $X^{(1)} = X$. For example, convolutional layers and fully-connected layers are typical linear layers. When $l < L$, the output features $\hat{Y}^{(l)} = lin^{(l)}(\theta^{(l)}, X^{(l)})$ is processed by an activation function $ac^{(l)}$ (e.g., the *ReLU* and *SoftMax* functions) that takes $\hat{Y}^{(l)}$ as input and outputs $X^{(l+1)}$, which is the input features of the $(l+1)$-th layer. Finally, $\hat{Y}^{(L)}$ is the score matrix for the given samples. Other classifiers may also fit this abstraction, e.g., logistic classifiers and SVM classifiers, which can be considered one-layer NNs.

Homomorphic encryption. HE is a cryptographic primitive that enables arithmetic operations over encrypted data. Given a plaintext pt, we denote its ciphertext under HE as $[\![pt]\!]$. A *fully HE* (FHE) system can support additions and multiplications between ciphertexts or between a ciphertext and a plaintext. A modern HE system, such as *CKKS* [5], usually provides the single instruction multiple data (SIMD) functionality: a ciphertext of CKKS has N *ciphertext slots* to store scalars, where N is a constant integer decided by the parameters of the HE system; it supports homomorphic entrywise addition $\oplus$ and multiplication (HMult) $\odot$ between two encrypted vectors (or between an encrypted vector and a plaintext vector), which are almost as efficient as additions and multiplications between two encrypted scalars (or between an encrypted scalar and a plaintext scalar), respectively; it can also rotate an encrypted vector $[\![pt]\!]$ j steps by a *homomorphic rotation* (HRot) $Rot([\![pt]\!], j)$.

As each layer of an NN is a linear function $lin^{(l)}$ of input features $X^{(l)}$ and model parameters $\theta^{(l)}$, we can homomorphically evaluate it by implementing additions $\oplus$ and multiplications $\odot$ between/with $[\![X^{(l)}]\!]$ and $[\![\theta^{(l)}]\!]$. Using SIMD, we can pack a batch of test samples as a matrix and simultaneously perform homomorphic operations over them.[1] However, exactly evaluating an activation function under HE is often difficult since it is usually nonlinear.

Secure matrix multiplication. Some types of linear layers involve matrix multiplications, which HE does not directly support. For example, a fully-connected layer is a matrix multiplication between a matrix of model parameters and a matrix of input features. To homomorphically evaluate a matrix multiplication, we need to transform it into a series of entrywise additions and multiplications that can be directly evaluated under HE.

Throughout this paper, when discussing secure matrix multiplication, we consider evaluating AB, where A is a $d_{out} \times d_{in}$ matrix of model parameters, and B is a $d_{in} \times m$ matrix of input features of m samples; d_{in} and d_{out} may be a linear layer's input and output sizes, respectively. For ease of discussion, we suppose $d_{out} \leq d_{in}$ without loss of generality.

Let us define some notations for matrix operations. Given a $d_1 \times d_2$ matrix $\mathcal{M}$, $\mathcal{M}[j, k]$ denotes the $((k-1) \bmod d_2 + 1)$-th entry of the $((j-1) \bmod d_1 + 1)$-th row of $\mathcal{M}$, and $\mathcal{M}[j_1 : j_2, k_1 : k_2]$ denotes the submatrix of $\mathcal{M}$ derived by extracting its $((k_1 - 1) \bmod d_2 + 1)$-th to $((k_2 - 1) \bmod d_2 + 1)$-th columns of its $((j_1 - 1) \bmod d_1 + 1)$-th to $((j_2 - 1) \bmod d_1 + 1)$-th rows. We use $\mathcal{M}_1; \mathcal{M}_2$ to denote a vertical concatenation of matrices $\mathcal{M}_1$ and $\mathcal{M}_2$ (if they have the same number of columns) and $\mathcal{M}_1 \big| \mathcal{M}_2$ to denote a horizontal concatenation of them (if they have the same number of rows). We also define four linear transformations σ, τ, ξ, and ψ that given a matrix $\mathcal{M}$ yield a transformed matrix of the same shape:

$$\begin{aligned} \forall j, k, \sigma(\mathcal{M})[j, k] = \mathcal{M}[j, j + k], \tau(\mathcal{M})[j, k] = \mathcal{M}[j + k, k], \\ \xi(\mathcal{M})[j, k] = \mathcal{M}[j, k + 1], \psi(\mathcal{M})[j, k] = \mathcal{M}[j + 1, k]. \end{aligned}$$

Additionally, when a superscript number o is assigned to a linear transformation, it means applying the transformation o times.

Additive secret sharing. ASS [61] protects a *secret* by splitting it into multiple *secret shares* such that they can be used to reconstruct the secret. In this paper, we only need to split a secret into two shares. Concretely, given a secret s in a finite field $\mathbb{Z}_p$, where p is a prime, we generate a uniformly random mask $r \in \mathbb{Z}_p$. Therefore, $s' = r$ and $s'' = (s - r) \bmod p$ are the two shares of s. To reconstruct the secret, we simply need to add up the shares in the field, i.e., $s \equiv (s' + s'') \bmod p$. For a real-number secret, we can encode it as an integer, split it into two integer secret shares, and finally decode them to derive two real-number shares. The method for implementing secret sharing on real numbers can be found in [59].

3 PROBLEM FORMULATION

System model. We study secure SV calculation in cross-silo FL. We consider the scenario where n organizations, i.e., clients $I = \{1, ..., n\}$, lack sufficient training data and join FL to train an accurate prediction model for their own use. They run T training rounds with the help of a parameter server.

Considering that clients' training data may vary in size, quality, and distribution, the server needs to evaluate their contributions to the accuracy of the final model θ^{T+1} after training. For fair evaluation, each client i contributes a test set of m_i samples $D_i = (X_i, Y_i)$, where X_i is a $d \times m_i$ feature matrix and Y_i is an m_i-sized

[1] A matrix's ciphertext is derived by horizontally scanning it into a vector.

vector of ground-truth labels; the k-th column of X_i and k-th entry of Y_i is the feature vector and ground-truth label of her k-th sample, respectively. The server evaluates the contributions based on a collective test set $\mathcal{D} = (D_1, ..., D_n)$, which has $M = \sum_{i=1}^{n} m_i$ samples.

However, calculating the original SV is impractical in FL. Intuitively, to calculate the SV, we must enumerate all subsets of clients, perform T rounds of FL for each subset to obtain a final model, and test all the final models. Retraining the models is significantly expensive for computation and communication [76], let alone securely training them; it may also cause extra privacy leakage owing to the more models to be released. Moreover, the SV assumes that the model utility is independent of the participation order of the clients, which does not stand true because clients may participate in or drop out from FL halfway [76].

Hence, we adopt the *Federated SV* (FSV) [76], a variant of the SV that addresses the aforementioned limitations well and guarantees the same advantageous properties as the SV. The FSV is based on the concept that a client's contribution to the final model θ^{T+1} is the sum of her contribution to each training round. Let θ_S^t denote the model aggregated from the local models of a set S of clients:

$$\theta_S^t = \begin{cases} \sum_{i \in S} \omega_{i|S}^t \cdot \theta_i^t & \text{if } S \neq \emptyset, \\ \theta^t & \text{if } S = \emptyset, \end{cases}$$

where $\omega_{i|S}^t \geq 0$ is an aggregation weight assigned to client i w.r.t. set S and determined by the training algorithm (e.g., FedAvg [44]). When $|S| > 1$, we term θ_S^t as an *aggregated model*. Then, for each round t, the server evaluates each client' *single-round SV* (SSV):

$$\phi_i^t = \begin{cases} \sum_{S \subseteq I^t \setminus \{i\}} \frac{|S|!(|I^t|-|S|-1)!}{|I^t|!} \big(v(\theta_{S \cup \{i\}}) - v(\theta_S) \big) & \text{if } i \in I^t, \\ 0 & \text{if } i \notin I^t, \end{cases}$$

where $v(\theta_S)$ is the prediction accuracy of model θ_S. Finally, the server aggregates each client i' SSVs into her FSV: $\phi_i = \sum_{t \in [T]} \phi_i^t$.

Threat model. Similar to prior works [37, 63, 82], we assume that all the parties are honest-but-curious and noncolluding because they are organizations complying with laws. Our problem is to design a protocol where the server coordinates the calculation of FSVs given encrypted local models while no party learns other parties' private information. We may include an auxiliary server to assist the principal server, which is a common model in secure computation literature. In this case, we reasonably assume that both servers, e.g., two cloud service providers, are honest-but-curious and will not collude with each other because collusion puts their business reputations at risk. Notably, a server is not a machine but a party that may possess multiple machines for parallel computing.

Privacy model. The private information considered in this study includes the test data and model parameters; the model structure decided by the clients is commonly assumed nonsensitive [27]. Readers may consider the test data as public information and question the need to protect them. Although a public test set is usually available to researchers for evaluating an ML algorithm or model, numerous practical cases exist where the test data are private:

- In collaborative ML marketplaces [17, 53, 65], clients submit their own test data as a specification of the model they want to collaboratively train and purchase.
- In federated evaluation [56, 75], Apple and Google let users compute some performance metrics of FL models on their own test sets to improve the user experience.
- For personalized cross-silo FL [9, 21], researchers assume that clients possess non-IID test data.

The test data may contain clients' private information and be their proprietary assets, so they intuitively need to be protected.

ALGORITHM 1: One-Server Protocol: HESV

```
 1: Server: Randomly select a leader client
 2: Leader: Generate a public key $pk$ and a private key $sk$ of HE and
    broadcast them among the other clients
 3: Each client $i$: Encrypt her test data and local models and upload them
 4: for $t$ in $\{1, ..., T\}$ do
 5:     for $S \subseteq I^t$ s.t. $|S| \geq 1$ do
 6:         Server: Compute $[\![\theta_S^t]\!]$ by aggregation under HE
 7:         Server: Run $\Pi_{HE}([\![\theta_S^t]\!], [\![X_i]\!], [\![Y_i]\!])$ to obtain $cnt_i$ for all $i \in I$
 8:         Server: Calculate $v(\theta_S^t) = (cnt_1 + ... + cnt_n)/M$
 9: Server: Calculate SSVs $\phi_1^t, ..., \phi_n^t, \forall t \in [T]$
10: Server: Calculate FSVs $\phi_1, ..., \phi_n$
```

ALGORITHM 2: Π_{HE}: Secure Testing for HESV

```
Input: encrypted model $[\![\theta]\!]$, features $[\![X]\!]$, and labels $[\![Y]\!]$
Output: count $cnt$ of correct predictions
1: for each layer $l \in \{1, ..., L\}$ of model $\theta$ do
2:     Server: Calculate $[\![\hat{Y}^{(l)}]\!] = lin^{(l)}([\![\theta^{(l)}]\!], [\![X^{(l)}]\!])$ and send $[\![\hat{Y}^{(l)}]\!]$
       to client $i_{l+1} \neq i_l$, where $[\![X^{(1)}]\!] = [\![X]\!]$
3:     Client $i_{l+1}$: Decrypt $[\![\hat{Y}^{(l)}]\!]$
4:     if $l < L$ then
5:         Client $i_{l+1}$: Compute $X^{(l+1)} = ac^{(l)}(\hat{Y}^{(l)})$, and upload $[\![X^{(l+1)}]\!]$
6:     else
7:         Client $i_{L+1}$: Compute $\hat{Y} = \textbf{Argmax}(\hat{Y}^{(L)})$, and upload $[\![\hat{Y}]\!]$
8: Server: Calculate $[\![\tilde{Y}]\!] = -1 \odot [\![\hat{Y}]\!] \oplus [\![Y]\!]$ and send it to client $i_{L+2}$
9: Client $i_{L+2}$: Decrypt $[\![\tilde{Y}]\!]$ and upload $cnt = \sum_{k=1}^{m} \mathbf{1}(|\tilde{Y}[k]| < 0.5)$
```

4 ONE-SERVER PROTOCOL: HESV

In this section, we present a one-server solution to secure SV calculation, named HESV (HE-Based Shapley Value).

4.1 Secure testing based purely on HE

HESV employs a purely HE-based privacy protection scheme that encrypts both model parameters and test data using HE, as described in Algorithm 1. To begin, each client encrypts her test data and local models using HE (Step 3). Then, for each training round t, the server enumerates all subsets of the selected clients I^t (Step 5); for each subset S, he aggregates the corresponding encrypted local models into $[\![\theta_S^t]\!]$ (Step 6), runs Algorithm 2 to count the correct predictions made by $[\![\theta_S^t]\!]$ (Step 7), and derives model utility $v(\theta_S^t)$ (Step 8). Finally, clients' SSVs and FSVs are computed based on the utilities of local and aggregate models (Steps 9 and 10).

Considering that HE supports nonlinear activations poorly, HESV adopts the *globally-encrypted-locally-decrypted* strategy [27, 83]: linear layers are homomorphically evaluated on the server's side, whereas activation functions are calculated on the clients' side without encryption. As depicted in Algorithm 2, for each layer l, there is a client i_l holding input features $X^{(l)}$. The server then evaluates the linear function $lin^{(l)}$ by applying c2c multiplications/additions between/with the encrypted input $[\![X^{(l)}]\!]$ (Step 2) and model parameters $[\![\theta^{(l)}]\!]$ and sends the output features $[\![\hat{Y}^{(l)}]\!]$ to client i_{l+1} for decryption (Step 3). Clients i_l and i_{l+1} should be different entities owing to a security issue that will be discussed in Section 6.1.3. If $l < L$, client i_{l+1} applies the activation function $ac^{(l)}(\hat{Y}^{(l)})$ to obtain the input $X^{(l+1)}$ of the subsequent layer (Step 5); otherwise, she calculates the predicted labels $\hat{Y}$ (Step 7). Finally, the server computes the differences $[\![\tilde{Y}]\!]$ between the predictions $[\![\hat{Y}]\!]$ and ground-truth labels $[\![Y]\!]$ and counts correct predictions with the help of some client i_{L+2} (Steps 8 and 9). Considering that HE introduces slight noise into ciphertexts, to tolerate the noise, we identify correct predictions by judging whether the absolute difference $|\tilde{Y}[k]|$ is smaller than 0.5 rather than whether $|\tilde{Y}[k]| = 0$.

4.2 Matrix Squaring

We propose an extension to the SOTA method called *Matrix Squaring* for homomorphic matrix multiplications under HESV.

4.2.1 SOTA method. When $d_{in} \leq \lfloor\sqrt{N}\rfloor$, the SOTA method [24] supports computing the matrix product AB under HE. Suppose that d_{out} exactly divides d_{in}.[2] This method evaluates AB as follows:

(1) **Squaring**: We obtain two square matrices $\bar{A}$ and $\bar{B}$ of order d_{in}. The $\bar{A}$ matrix vertically packs d_{in}/d_{out} copies of A, i.e., $\bar{A} = (A; \ldots; A)$, while $\bar{B}$ is derived by padding $(d_{in} - m)$ zero-valued columns $\mathbf{0}_{d_{in}\times(d_{in}-m)}$ to the end edge of B, i.e., $\bar{B} = (B \,|\, \mathbf{0}_{d_{in}\times(d_{in}-m)})$.

(2) **Linear transformation**: We linearly transform $\bar{A}$ and $\bar{B}$ into two sets of matrices $\{\bar{A}^{(o)}\}_{o=1}^{d_{out}}$ and $\{\bar{B}^{(o)}\}_{o=1}^{d_{out}}$, respectively. Matrices $\bar{A}^{(1)}$ and $\bar{B}^{(1)}$ are derived by rotating the j-th row of $\bar{A}$ $j-1$ steps for all $j \in [d_{in}]$ and rotating the k-th column of $\bar{B}$ $k-1$ steps for all $k \in [d_{in}]$, respectively, i.e., $\bar{A}^{(1)} = \sigma(\bar{A})$, and $\bar{B}^{(1)} = \tau(\bar{B})$. Then, for each $o \in [2, d_{out}]$, we can shift $\bar{A}^{(1)}$ $o-1$ columns and $\bar{B}^{(1)}$ $o-1$ rows to obtain the $\bar{A}^{(o)}$ and $\bar{B}^{(o)}$ matrices, i.e., $\bar{A}^{(o)} = \xi^{(o-1)}(\bar{A}^{(1)})$, and $\bar{B}^{(o)} = \psi^{(o-1)}(\bar{B}^{(1)})$.

(3) **Encryption**: We encrypt the transformed matrices $\{\bar{A}^{(o)}\}_{o=1}^{d_{out}}$ and $\{\bar{B}^{(o)}\}_{o=1}^{d_{out}}$ and upload them to the server.

(4) **Entrywise operations**: The server computes $[\![H]\!] = [\![\bar{A}^{(1)}]\!] \odot [\![\bar{B}^{(1)}]\!] \oplus \ldots \oplus [\![\bar{A}^{(d_{out})}]\!] \odot [\![\bar{B}^{(d_{out})}]\!]$.

(5) **Rotation and extraction**: The matrix product AB can be obtained by vertically splitting matrix H into d_{in}/d_{out} submatrices, adding them up, and extracting the first m columns of the result, i.e., $AB = \tilde{H}[1 : d_{out}, 1 : m]$, where $[\![\tilde{H}]\!] = \oplus_{o=0}^{d_{in}/d_{out}-1} Rot([\![H]\!], d_{out}\cdot d_{in}\cdot o)$.[3]

However, a ciphertext of HE does not have sufficient slots to store a large matrix of order $d_{in} > \lfloor\sqrt{N}\rfloor$. This is a typical case because an NN's input is usually large. Even if we can use multiple ciphertexts to store the matrix, slot rotations across ciphertexts are not supported, which makes the SOTA method fail.

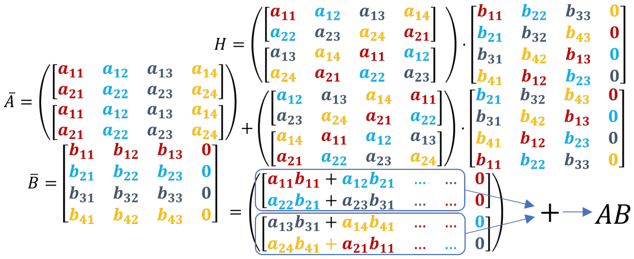

$H =$
a_{11} a_{12} a_{13} a_{14}
a_{22} a_{23} a_{24} a_{21}
a_{13} a_{14} a_{11} a_{12}
a_{24} a_{21} a_{22} a_{23}
b_{11} b_{22} b_{33} 0
b_{21} b_{32} b_{43} 0
b_{31} b_{42} b_{13} 0
b_{41} b_{12} b_{23} 0
$\bar{A} =$
a_{11} a_{12} a_{13} a_{14}
a_{21} a_{22} a_{23} a_{24}
a_{11} a_{12} a_{13} a_{14}
a_{21} a_{22} a_{23} a_{24}
+
a_{12} a_{13} a_{14} a_{11}
a_{23} a_{24} a_{21} a_{22}
a_{14} a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23} a_{24}
b_{21} b_{32} b_{43} 0
b_{31} b_{42} b_{13} 0
b_{41} b_{12} b_{23} 0
b_{11} b_{22} b_{33} 0
$\bar{B} =$
b_{11} b_{12} b_{13} 0
b_{21} b_{22} b_{23} 0
b_{31} b_{32} b_{33} 0
b_{41} b_{42} b_{43} 0
=
$a_{11}b_{11} + a_{12}b_{21}$ … … 0
$a_{22}b_{21} + a_{23}b_{31}$ … … 0
$a_{13}b_{31} + a_{14}b_{41}$ … … 0
$a_{24}b_{41} + a_{21}b_{11}$ … … 0
+
AB

Figure 2: SOTA method for matrix multiplication.

Example 4.1. Figure 2 shows how the SOTA method works for a 2×4 matrix A and a 4×3 matrix B with $N = 16$, where $A[i, j] = a_{ij}$, and $B[i, j] = b_{ij}$. First, we vertically pack two copies of A and pad B with zeros to derive 4×4 matrices $\bar{A}$ and $\bar{B}$, respectively. We then apply entrywise operations over two pairs of transformed matrices to obtain H. Essentially, AB is derived by adding $H[1 : 2, 1 : 3]$ and $H[3 : 4, 1 : 3]$. However, when $N = 12$, the transformed matrices are overly large to encrypt into a ciphertext.

4.2.2 Our improvement. To address this issue, Matrix Squaring involves dividing matrices A and B into smaller submatrices that can be stored in a ciphertext. Suppose that $\lfloor\sqrt{N}\rfloor$ exactly divides d_{in} without loss of generality.[4] Concretely, when $d_{in} > \lfloor\sqrt{N}\rfloor$, we can vertically split A every $\lfloor\sqrt{N}\rfloor$-th column to obtain K submatrices $A_{(\cdot,1)}, \ldots, A_{(\cdot,K)}$ and horizontally split B every $\lfloor\sqrt{N}\rfloor$-th row to derive K submatrices $B_{(1,\cdot)}, \ldots, B_{(K,\cdot)}$, where $K = d_{in}/\lfloor\sqrt{N}\rfloor$, $(A_{(\cdot,1)} \,|\, \ldots \,|\, A_{(\cdot,K)}) = A$, and $(B_{(1,\cdot)}; \ldots; B_{(K,\cdot)}) = B$. Then, we have

$$AB = \sum_{k=1}^{K} A_{(\cdot,k)} B_{(k,\cdot)}.$$

Therefore, we can evaluate AB under HE by applying the SOTA method over $d_{in}/\lfloor\sqrt{N}\rfloor$ pairs of submatrices $A_{(\cdot,k)}$ and $B_{(k,\cdot)}$ and aggregating the results. This inherently requires that m should not exceed $\lfloor\sqrt{N}\rfloor$; otherwise, any square matrix transformed from a $\lfloor\sqrt{N}\rfloor\times m$ matrix $B_{(k,\cdot)}$ cannot be encrypted into a single ciphertext. Furthermore, when $d_{out} > \lfloor\sqrt{N}\rfloor$, we can horizontally split $A_{(\cdot,k)}$ every $\lfloor\sqrt{N}\rfloor$-th row into $A_{(1,k)}, \ldots, A_{(J,k)}$, where $J = \lceil d_{out}/\lfloor\sqrt{N}\rfloor\rceil$, and $(A_{(1,k)}; \ldots; A_{(J,k)}) = A_{(\cdot,k)}$. Then, we have

$$A_{(\cdot,k)} B_{(k,\cdot)} = (A_{(1,k)} B_{(k,\cdot)}; \ldots; A_{(J,k)} B_{(k,\cdot)}).$$

Hence, we can evaluate $A_{(\cdot,k)} B_{(k,\cdot)}$ by applying the SOTA method over J pairs of $A_{(j,k)}$ and $B_{(k,\cdot)}$ and vertically packing the results.

5 TWO-SERVER PROTOCOL: SECSV

HESV has three considerable drawbacks. First, it involves numerous c2c multiplications, which are highly inefficient in computation. Second, it cannot fully utilize the SIMD feature of HE. Since clients encrypt test samples on their local sides, the server cannot pack samples from different sources, which may waste some ciphertext slots. Third, it fully evaluates the entire test set for all aggregated models, which is time-consuming. In this section, we propose a two-server protocol with an auxiliary server named SecSV (Secure

[2] If d_{out} divides d_{in} with a remainder, we can pad A with zero-valued rows to obtain a $d'_{out} \times d_{in}$ matrix such that d'_{out} exactly divides d_{in}.

[3] For computing $[\![\tilde{H}]\!]$, the server can apply a repeated doubling approach to improve efficiency [24].

[4] We can pad A and B with zeros to ensure this condition.

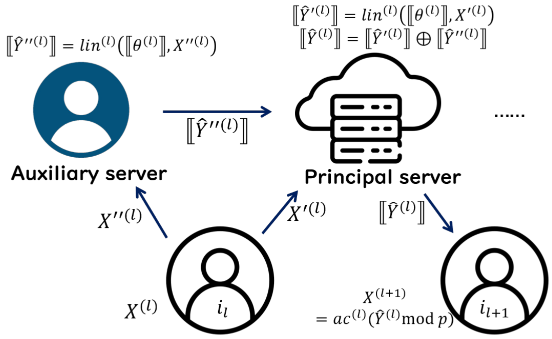

$[\![\hat{Y}''^{(l)}]\!] = lin^{(l)}([\![\theta^{(l)}]\!], X''^{(l)})$
$[\![\hat{Y}'^{(l)}]\!] = lin^{(l)}([\![\theta^{(l)}]\!], X'^{(l)})$
$[\![\hat{Y}^{(l)}]\!] = [\![\hat{Y}'^{(l)}]\!] \oplus [\![\hat{Y}''^{(l)}]\!]$
$[\![\hat{Y}''^{(l)}]\!]$
......
Auxiliary server
Principal server
$X''^{(l)}$
$X'^{(l)}$
$[\![\hat{Y}^{(l)}]\!]$
$X^{(l)}$
i_l
$X^{(l+1)} = ac^{(l)}(\hat{Y}^{(l)} \bmod p)$
i_{l+1}

Figure 3: Secure testing for SecSV.

Shapley Value) to overcome the drawbacks of HESV. The features of this protocol are (1) a hybrid secure testing scheme, (2) an efficient homomorphic matrix multiplication method, and (3) an acceleration technique for SV calculation.

5.1 Hybrid Secure Testing Scheme

SecSV adopts a hybrid scheme for secure testing: it encrypts models by HE but protects test data by ASS. An auxiliary server $\mathcal{A}$ is needed to help the principal server $\mathcal{P}$ test models on secretly shared test data. Concretely, as shown in Algorithm 3, each client i encrypts her local models by HE but protects her test data D_i by splitting it into two secret shares D'_i and D''_i (Step 3). Thereafter, the two servers evaluate the shares $\mathcal{D}', \mathcal{D}''$ of the collective test set $\mathcal{D} = (D_1, ..., D_n)$ by running Algorithm 4 (Step 10). Figure 3 shows that because the shares $X'^{(l)}, X''^{(l)}$ of input features $X^{(l)}$ are in plaintext form for all layers l, c2c multiplications are avoided.

Algorithm 4 shows how SecSV evaluates an encrypted model. For each model layer l, server $\mathcal{P}$ holds a share $X'^{(l)}$ of the input features $X^{(l)}$ while server $\mathcal{A}$ possesses the other share $X''^{(l)}$. They each evaluate the linear function $lin^{(l)}$ over their own shares to compute shares $[\![\hat{Y}'^{(l)}]\!]$ and $[\![\hat{Y}''^{(l)}]\!]$, respectively (Steps 4 and 5). Then, after receiving $[\![\hat{Y}''^{(l)}]\!]$ from server $\mathcal{A}$, server $\mathcal{P}$ adds up $[\![\hat{Y}'^{(l)}]\!]$ and $[\![\hat{Y}''^{(l)}]\!]$ to reconstruct the output features $[\![\hat{Y}^{(l)}]\!]$ (Step 6), which is sent to client i_{l+1} for decryption and modulo (Step 7). If $l < L$, client i_{l+1} activates the output features and generates shares $X'^{(l+1)}, X''^{(l+1)}$ of the activated features $X^{(l+1)}$ for evaluating the next layer (Step 9); otherwise, client i_{L+1} computes the predicted labels $\hat{Y}$ and generates shares $\hat{Y}', \hat{Y}''$ for comparison with the shares Y', Y'' of the ground-truth labels Y (Step 11). After obtaining the absolute differences $\tilde{Y}$ between $\hat{Y}$ and Y (Steps 12 and 13), server $\mathcal{P}$ updates an ID set Φ that contains the IDs of the correctly predicted samples with a tolerable difference $\tilde{Y}[k] < 0.5$ (Step 14).

Example 5.1. Consider 4 clients and 3-layer models. When testing client 1' local model θ^t_1, given shares $X'^{(1)}, X''^{(1)}$ of input features from client 2, the servers evaluate layer 1 under HE, aggregate shares $\hat{Y}'^{(1)}, \hat{Y}''^{(1)}$ of output features, and return $\hat{Y}^{(1)}$ to client 3 for activation. Similarly, given $X'^{(2)}, X''^{(2)}$ from client 3, the servers evaluate layer 2 and send $\hat{Y}^{(2)}$ to client 4. Finally, client 3 obtains the output features $\hat{Y}^{(3)}$, computes the predicted labels $\hat{Y}$, and returns shares $\hat{Y}', \hat{Y}''$ to the servers for comparison with the shares Y', Y'' of ground-truth labels.

ALGORITHM 3: SecSV

```
1: Server $\mathcal{P}$: Randomly select a leader client
2: Leader: Generate a public key $pk$ and a private key $sk$ of HE and
   broadcast them among the other clients
3: Each client $i$: Encrypt her own local models and send them to the two
   servers; then, generate secret shares $D'_i, D''_i$ of $D_i$, send $D'_i$ to server
   $\mathcal{P}$, and send $D''_i$ to server $\mathcal{A}$
4: for $t$ in $\{1, ..., T\}$ do
5:     if skipSamples == True then
6:         Server $\mathcal{P}$: Run SampleSkip($\{[\![\theta^t_i]\!]\}_{\forall i \in I^t}, (\mathcal{D}', \mathcal{D}'')$) to obtain
           utilities $\{v(\theta^t_S)\}_{\forall S \subseteq I^t, |S|>0}$
7:     else
8:         for $S \subseteq I^t, |S| > 0$ do
9:             Servers $\mathcal{P}, \mathcal{A}$: Compute $[\![\theta^t_S]\!]$ by aggregation under HE
10:            Server $\mathcal{P}$: Run $\Pi_{Sec}([\![\theta^t_S]\!], (\mathcal{D}', \mathcal{D}''))$ to obtain $\Phi^t_S$
11:            Server $\mathcal{P}$: Calculate $v(\theta^t_S) = |\Phi^t_S|/M$
12: Server $\mathcal{P}$: Calculate SSVs $\phi^t_1, ..., \phi^t_n, \forall t \in [T]$
13: Server $\mathcal{P}$: Calculate FSVs $\phi_1, ..., \phi_n$
```

ALGORITHM 4: Π_{Sec}: Secure Testing for SecSV

```
Input: encrypted model $[\![\theta]\!]$, and secret shares $\mathcal{D}', \mathcal{D}''$ of $\mathcal{D}$
Output: IDs $\Phi$ of correctly predicted samples
1: $\Phi \leftarrow \emptyset$
2: for each $D' = (X', Y'), D'' = (X'', Y'') \in (\mathcal{D}', \mathcal{D}'')$ do
3:     for each model layer $l \in \{1, ..., L\}$ do
4:         Server $\mathcal{A}$: Calculate $[\![\hat{Y}''^{(l)}]\!] = lin^{(l)}([\![\theta^{(l)}]\!], X''^{(l)})$ and send
           $[\![\hat{Y}''^{(l)}]\!]$ to server $\mathcal{P}$, where $X''^{(1)} = X''$
5:         Server $\mathcal{P}$: Calculate $[\![\hat{Y}'^{(l)}]\!] = lin^{(l)}([\![\theta^{(l)}]\!], X'^{(l)})$, where
           $X'^{(1)} = X'$
6:         Server $\mathcal{P}$: Send$[\![\hat{Y}^{(l)}]\!] = [\![\hat{Y}'^{(l)}]\!] \oplus [\![\hat{Y}''^{(l)}]\!]$ to client $i_{l+1} \neq i_l$
7:         Client $i_{l+1}$: Compute $\hat{Y}^{(l)} = Decrypt([\![\hat{Y}^{(l)}]\!]) \bmod p$
8:         if $l < L$ then
9:             Client $i_{l+1}$: Compute $X^{(l+1)} = ac^{(l)}(\hat{Y}^{(l)})$, generate shares
               $X'^{(l+1)}, X''^{(l+1)}$, and distribute them to servers $\mathcal{P}$ and $\mathcal{A}$
10:        else
11:            Client $i_{L+1}$: Calculate $\hat{Y} = \mathbf{Argmax}(\hat{Y}^{(L)})$, generate shares
               $\hat{Y}', \hat{Y}''$, and distribute them to servers $\mathcal{P}$ and $\mathcal{A}$
12:    Server $\mathcal{A}$: Compute and send $\tilde{Y}'' = \hat{Y}'' - Y''$ to server $\mathcal{P}$
13:    Server $\mathcal{P}$: Calculate $\tilde{Y} = abs((\hat{Y}' - Y' + \tilde{Y}'') \bmod p)$, where $abs$
       denotes taking the entrywise absolute value
14:    Server $\mathcal{P}$: Update the IDs of correctly predicted samples
       $\Phi \leftarrow \Phi \cup \{id(D[k]) | \tilde{Y}[k] < 0.5\}$, where $id(D[k])$ is the ID of
       test sample $D[k]$
15: return $\Phi$
```

5.2 Matrix Reducing

Our insight. We then propose a secure matrix multiplication method named *Matrix Reducing*. To evaluate AB under HE, we have to transform A and B such that they have the same shape because HE only allows entrywise operations. For example, the SOTA method squares and linearly transforms A and B to derive pairs of square matrices, applies entrywise multiplications between each pair of matrices, and aggregates the multiplication results. However, the aggregated square matrix H (see Section 4.2.1) is not the matrix product AB, and the SOTA method further needs to rotate H to

ALGORITHM 5: Matrix Reducing

```
Input: $d_{out} \times d_{in}$ matrix $A$, and $d_{in} \times m$ matrix $B$ ($m \le \lfloor N/d_{out} \rfloor$)
Output: encrypted matrix product $[\![AB]\!]$
Client: Horizontally pack $\lceil m/d_{in} \rceil$ copies of $A$ into $\bar{A} = (A \,|\, \dots \,|\, A)$
Client: Run the first two steps of $\delta(\bar{A}, B)$ to obtain $\{\tilde{A}^{(o)}, \tilde{B}^{(o)}\}_{o=1}^{d_{in}}$
Client: Encrypt $\{\tilde{A}^{(o)}\}_{o=1}^{d_{in}}$ and $\{\tilde{B}^{(o)}\}_{o=1}^{d_{in}}$ and upload them
Server: Return $[\![AB]\!] = [\![\tilde{A}^{(1)}]\!] \odot [\![\tilde{B}^{(1)}]\!] \oplus \dots \oplus [\![\tilde{A}^{(d_{in})}]\!] \odot [\![\tilde{B}^{(d_{in})}]\!]$
```

aggregate its submatrices, which is expensive under HE. This inefficiency results from the need to square A and B: the SOTA method squares a rectangular A by packing multiple copies of A together, which enables parallel computation over A and avoids the waste of some ciphertext slots.

Considering that the matrix product AB is a $d_{out} \times m$ matrix, we introduce the concept of transforming A and B into $d_{out} \times m$ matrices to avoid homomorphic rotations. Concretely, in the unencrypted environment, to obtain each entry in the matrix product AB, we need d_{in} multiplications between entries in A and B; this suggests that we can somehow preprocess A and B to generate d_{in} pairs of $d_{out} \times m$ matrices, apply entrywise multiplications between each pair of matrices, and aggregate the multiplication results to derive AB. We design Matrix Reducing based on this idea.

Design details. To assist in understanding Matrix Reducing, we introduce a function $R = \delta(\bar{A}, \bar{B})$ that takes as input a $d_{out} \times d$ matrix $\bar{A}$ and a $d_{in} \times m$ matrix $\bar{B}$ and outputs a $d_{out} \times m$ matrix R, where $d \ge m$ and d_{in} exactly divides d. $\delta(\bar{A}, \bar{B})$ runs as follows:

(1) **Linear transformation:** We linearly transform $\bar{A}$ and $\bar{B}$ into two sets of matrices $\{\bar{A}^{(o)}\}_{o=1}^{d_{in}}$ and $\{\bar{B}^{(o)}\}_{o=1}^{d_{in}}$, where $\bar{A}^{(o)} = \xi^{(o-1)}(\sigma(\bar{A}))$, and $\bar{B}^{(o)} = \psi^{(o-1)}(\tau(\bar{B}))$.

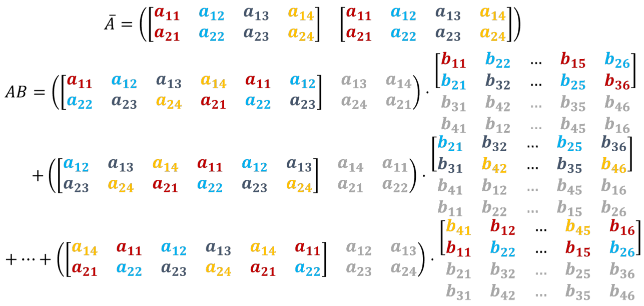
(2) **Reduction:** For each $o \in [d_{in}]$, we extract two $d_{out} \times m$ matrices $\tilde{A}^{(o)}$ and $\tilde{B}^{(o)}$ from $\bar{A}^{(o)}$ and $\bar{B}^{(o)}$, respectively, where $\tilde{A}^{(o)} = \bar{A}^{(o)}[1 : d_{out}, 1 : m]$, and $\tilde{B}^{(o)} = \bar{B}^{(o)}[1 : d_{out}, 1 : m]$.

(3) **Element-wise operations:** We apply entrywise multiplications and additions to compute $R = \sum_{o=1}^{d_{in}} \tilde{A}^{(o)} \cdot \tilde{B}^{(o)}$.

When $m \le d_{in}$, according to lemma 5.2, we have $AB = \delta(A, B)$. Therefore, we can encrypt matrices $\{\tilde{A}^{(o)}\}_{o=1}^{d_{in}}$ and $\{\tilde{B}^{(o)}\}_{o=1}^{d_{in}}$ and apply homomorphic entrywise multiplications and additions over them to compute $[\![AB]\!]$.

Lemma 5.2 (Correctness). *We have $AB = \delta(A, B)$ for any $d_{out} \times d_{in}$ matrix A and $d_{in} \times m$ matrix B, where $m \le d_{in}$, and $d_{out} \le d_{in}$.*[5]

Then, when $m \in (d_{in}, \lfloor N/d_{out} \rfloor]$, Matrix Reducing evaluates AB by horizontally packing multiple copies of A into a $d_{out} \times m$ matrix. Suppose that d_{in} exactly divides m when $m > d_{in}$ for ease of discussion. The matrix product AB can be vertically split into $d_{out} \times d_{in}$ matrices $\{AB_{(\cdot,o)}\}_{o=1}^{m/d_{in}}$, i.e., $AB = \left(AB_{(\cdot,1)} \,\middle|\, \dots \,\middle|\, AB_{(\cdot,m/d_{in})}\right)$, where $B_{(\cdot,o)} = B\left[1 : d_{in}, (o-1) \cdot d_{in} + 1 : o \cdot d_{in}\right], \forall o \in [m/d_{in}]$. According to lemma 5.2, we have $AB = \left(\delta(A, B_{(\cdot,1)}) \,\middle|\, \dots \,\middle|\, \delta(A, B_{(\cdot,m/d_{in})})\right)$. Then, according to Lemma 5.3, we can conclude that $AB = \delta(\bar{A}, B)$, where $\bar{A} = (A \,|\, \dots \,|\, A)$ is a $d_{out} \times m$ matrix that contains m/d_{in} copies of A. Hence, as shown in Algorithm 5, Matrix Reducing evaluates $[\![AB]\!]$ by homomorphically computing $\delta(\bar{A}, B)$, which involves only d_{in} homomorphic multiplications and no homomorphic rotation.



$\bar{A} = \left(\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix}\right)$
$AB = \left(\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{11} & a_{12} \\ a_{22} & a_{23} & a_{24} & a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{matrix} a_{13} & a_{14} \\ a_{24} & a_{21} \end{matrix}\right) \cdot \begin{matrix} b_{11} & b_{22} & \dots & b_{15} & b_{26} \\ b_{21} & b_{32} & \dots & b_{25} & b_{36} \\ b_{31} & b_{42} & \dots & b_{35} & b_{46} \\ b_{41} & b_{12} & \dots & b_{45} & b_{16} \end{matrix}$
$+ \left(\begin{bmatrix} a_{12} & a_{13} & a_{14} & a_{11} & a_{12} & a_{13} \\ a_{23} & a_{24} & a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{matrix} a_{14} & a_{11} \\ a_{21} & a_{22} \end{matrix}\right) \cdot \begin{matrix} b_{21} & b_{32} & \dots & b_{25} & b_{36} \\ b_{31} & b_{42} & \dots & b_{35} & b_{46} \\ b_{41} & b_{12} & \dots & b_{45} & b_{16} \\ b_{11} & b_{22} & \dots & b_{15} & b_{26} \end{matrix}$
$+ \dots + \left(\begin{bmatrix} a_{14} & a_{11} & a_{12} & a_{13} & a_{14} & a_{11} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{21} & a_{22} \end{bmatrix} \begin{matrix} a_{12} & a_{13} \\ a_{23} & a_{24} \end{matrix}\right) \cdot \begin{matrix} b_{41} & b_{12} & \dots & b_{45} & b_{16} \\ b_{11} & b_{22} & \dots & b_{15} & b_{26} \\ b_{21} & b_{32} & \dots & b_{25} & b_{36} \\ b_{31} & b_{42} & \dots & b_{35} & b_{46} \end{matrix}$

Figure 4: Matrix Reducing for matrix multiplication.

Lemma 5.3 (Composition). *Given any $d_{out} \times d_{in}$ matrix A, $d_{in} \times d_{in}$ matrix B_1, and $d_{in} \times m'$ matrix B_2, where $d_{in} \ge m'$, we have*

$$(\delta(A, B_1) \,|\, \delta(A, B_2)) = \delta((A \,|\, A), (B_1 \,|\, B_2))$$

Example 5.4. Figure 4 depicts how Matrix Reducing works for a 2 × 4 matrix A and a 4 × 6 matrix B, where $A[i, j] = a_{ij}$, and $B[i, j] = b_{ij}$. First, we horizontally pack two copies of A to derive a 2 × 8 matrix $\bar{A}$. Then, we linearly transform $\bar{A}$ and B and reduce the transformed matrices into 2 × 6 matrices. Finally, we apply entrywise operations over the reduced matrices to derive AB.

5.3 SampleSkip

Finally, we propose to further accelerate secure SV calculation by reducing the number of test samples for secure model testing. Concretely, if some test samples are highly likely to be correctly predicted by a model, which we refer to as *skippable samples*, we can skip these samples during model testing to save time. We then identify skippable samples: *If a sample can be correctly predicted by two models, it can also be correctly predicted by their aggregation, and thus, it is skippable; otherwise, it is a discriminative sample that distinguishes the models' utilities and the clients' contributions.*

Hence, we propose *SampleSkip* to speed up secure SV calculation by skipping test samples. Let Φ_S^t denote an ID set that contains the IDs of the test samples that can be correctly predicted by model θ_S^t, where $S \subseteq I^t$ is a subset of clients. As shown in Algorithm 6, in each round t, server $\mathcal{P}$ first initializes a dictionary $\mathbf{\Phi}^t$ to record Φ_S^t for all subsets $S \subseteq I^t$ (Step 1). For each local model $\theta_i^t, i \in I^t$, the servers test it on the entire test set $(\mathcal{D}', \mathcal{D}'')$ and record $\Phi_{\{i\}}^t$ (Step 3). Then, for each aggregated model $\theta_S^t, |S| \ge 2$, the servers drop the skippable samples Ψ_S^t from $(\mathcal{D}', \mathcal{D}'')$ and test the model on the discriminative samples $(\mathcal{D}'_{-\Psi_S^t}, \mathcal{D}''_{-\Psi_S^t})$ (Steps 8-10); the skippable samples Ψ_S^t are counted as correctly predicted samples (Step 11). Note that SampleSkip also applies to nonsecure SV calculation.

We also propose Algorithm 7 that identifies skippable samples for aggregated models. Given that an aggregated model θ_S^t can be aggregated from multiple pairs of models, we need to find the union of the skippable samples determined by each pair of models. Hence, Algorithm 7 enumerates all possible pairs of nonempty subsets $(S', S \setminus S')$ of the given set S, identifies skippable samples for each

[5] See Appendix B to find all the missing proofs.

ALGORITHM 6: SampleSkip

```
Input: encrypted models $\{[\![\theta_i^t]\!]\}_{\forall i \in I^t}$, secret shares $\mathcal{D}', \mathcal{D}''$ of $\mathcal{D}$
Output: model utilities $\{v(\theta_S^t)\}_{\forall S \subseteq I^t, |S|>0}$
Server $\mathcal{P}$: Initialize a dictionary $\mathbf{\Phi}^t$.
for $i \in I^t$ do
    Server $\mathcal{P}$: Run $\Pi_{Sec}([\![\theta_i^t]\!], (\mathcal{D}', \mathcal{D}''))$ to obtain $\Phi_{\{i\}}^t$
    Server $\mathcal{P}$: Calculate $v(\theta_i^t) = |\Phi_{\{i\}}^t|/M$
for $j = 2$ to $j = |I^t|$ do
    for $S \subseteq I^t, |S| = j$ do
        Servers $\mathcal{P}, \mathcal{A}$: Calculate $[\![\theta_S^t]\!]$ under HE
        Server $\mathcal{P}$: Compute IDs $\Psi_S^t = \text{FindSkippable}(S, \mathbf{\Phi}^t)$.
        Servers $\mathcal{P}, \mathcal{A}$: Drop skippable samples $\Psi_S^t$ from $(\mathcal{D}', \mathcal{D}'')$ to
        obtain discriminative samples $(\mathcal{D}'_{-\Psi_S^t}, \mathcal{D}''_{-\Psi_S^t})$
        Server $\mathcal{P}$: Run $\Pi_{Sec}([\![\theta_S^t]\!], (\mathcal{D}'_{-\Psi_S^t}, \mathcal{D}''_{-\Psi_S^t}))$ to obtain $\Phi_S^t$
        Server $\mathcal{P}$: $\Phi_S^t \leftarrow \Phi_S^t \cup \Psi_S^t$
        Server $\mathcal{P}$: Calculate $v(\theta_S^t) = |\Phi_S^t|/M$
return $\{v(\theta_S^t)\}_{\forall S \subseteq I^t, |S|>0}$
```

ALGORITHM 7: FindSkippable

```
Input: a set $S$ of clients, a dictionary $\Phi^t$
Output: IDs $\Psi_S^t$ of skippable samples for model $\theta_S^t$
$\Psi_S^t \leftarrow \emptyset$
for each pair of nonempty subsets $(S', S \setminus S')$ of $S$ do
    if $\Phi_{S'}^t, \Phi_{S \setminus S'}^t$ exist then
        $\Psi_S^t \leftarrow \Psi_S^t \cup (\Phi_{S'}^t \cap \Phi_{S \setminus S'}^t)$
return $\Psi_S^t$
```

pair, and returns the union set of skippable samples; this suggests that SampleSkip should iterate over all the subsets $S \subseteq I^t, |S| > 1$ in ascending order of their sizes such that $\Phi_{S'}^t$ and $\Phi_{S \setminus S'}^t$ are already recorded in Φ^t for identifying Ψ_S^t. Notably, although this algorithm takes $O(2^{|S|})$ complexity, it hardly burdens SecSV because the time for running it is far less than that for evaluating models under HE.

Example 5.5. Consider 3 clients and 4 test samples $x_1, ..., x_4$. If models $\theta_{\{1\}}^t, \theta_{\{2,3\}}^t$ correctly predict x_1, x_2, models $\theta_{\{2\}}^t, \theta_{\{1,3\}}^t$ correctly predict x_2, x_3, and models $\theta_{\{3\}}^t, \theta_{\{1,2\}}^t$ correctly predict x_3, x_4, then the servers can skip all the test samples when testing $\theta_{\{1,2,3\}}^t$.

6 THEORETICAL ANALYSIS AND DISCUSSION

6.1 Theoretical Analysis

6.1.1 Time cost. Table 1 summarizes the time complexities of Matrix Squaring and Matrix Reducing. Note that the batch size m is a hyperparameter set by the server(s); the requirements on m suggest the maximum batch size such that a batch of samples can be stored in a ciphertext, which maximizes computation efficiency. Considering that the maximum batch size may differ between the two methods, the complexities should be averaged over the maximum batch sizes for a fair comparison. Evidently, Matrix Reducing outperforms Matrix Squaring. First, unlike Matrix Squaring, Matrix Reducing does not require any homomorphic rotation. Regarding homomorphic multiplications, Matrix Squaring and Matrix Reducing take $O(d_{in} \cdot d_{out}/\sqrt{N})$ and $O(d_{in})$ complexity for evaluating $m \leq \min\{d_{in}, \lfloor\sqrt{N}\rfloor\}$ and $m \leq \lfloor N/d_{out} \rfloor$ samples, respectively; thus, when we batch $\min\{d_{in}, \lfloor\sqrt{N}\rfloor\}$ samples for the former method and $\lfloor N/d_{out} \rfloor$ samples for the latter, the latter's complexity $O(d_{in} \cdot d_{out}/N)$ for each sample is better than the former's complexity $O(d_{in} \cdot d_{out}/\sqrt{N}/\min\{d_{in}, \lfloor\sqrt{N}\rfloor\})$. The uses of the different matrix multiplication methods result in the different time costs of HESV and SecSV, as characterized below.

Table 1: Comparison of time cost for evaluating AB.

	Matrix Squaring	Matrix Reducing
Batch size m	$m \leq \min\{d_{in}, \lfloor\sqrt{N}\rfloor\}$	$m \leq \lfloor N/d_{out} \rfloor$
Complexity of HMult	$O(d_{in} \cdot d_{out}/\sqrt{N})$	$O(d_{in})$
Complexity of HRot	$O(d_{in}/(d_{out} \bmod \sqrt{N}))$	0

LEMMA 6.1. *Consider models with L layers of matrix multiplication where the input and output sizes of layer l are $d_{in}^{(l)}$ and $d_{out}^{(l)}$, respectively. For secure SV calculation, HESV needs $O(\frac{2^n \cdot M \cdot T \cdot (\sum_{l=1}^{L} d_{in}^{(l)} \cdot d_{out}^{(l)})}{\min\{d_{in}^{(1)}, ..., d_{in}^{(L)}, \sqrt{N}\} \cdot \sqrt{N}})$ HMults and $O(\frac{2^n \cdot M \cdot T \cdot \sum_{l=1}^{L} \frac{d_{in}}{d_{out} \bmod \sqrt{N}}}{\min\{d_{in}^{(1)}, ..., d_{in}^{(L)}, \sqrt{N}\}})$ HRots, while SecSV needs $O(\frac{2^n \cdot M \cdot T \cdot \max\{d_{out}^{(1)}, ..., d_{out}^{(L)}\} \sum_{l=1}^{L} d_{in}^{(l)}}{N})$ HMults and no HRot.*

6.1.2 Error of SampleSkip. According to Theorem 6.3, for *linear classifiers*, e.g., the widely-used logistic classifiers and linear SVM classifiers, the intuition behind SampleSkip holds.

Definition 6.2 (Linear classifier). A linear classifier $f : R^d \rightarrow R^c$ is of the form $f_\theta(x) = \eta(\theta^w x + \theta^b)$, where model $\theta = (\theta^w \,|\, \theta^b)$ consists of a $c \times d$ matrix θ^w of weights and a c-sized vector θ^b of biases, and $\eta : R^c \rightarrow R^c$ can be any entrywise strictly increasing function: given any c-sized vector v, for any $j_1, j_2 \in [c]$, if $v[j_1] > v[j_2]$, we have $\eta(v)[j_1] > \eta(v)[j_2]$.

THEOREM 6.3. *For any linear classifier f, any test sample (x, y), and any two models $\theta_{S_1}, \theta_{S_2}$, if they satisfy* $\arg\max_j f_{\theta_{S_1}}(x)[j] = \arg\max_j f_{\theta_{S_2}}(x)[j] = y$, *then for any* $\theta_{S_1 \cup S_2} = \omega_{S_1} \cdot \theta_{S_1} + \omega_{S_2} \cdot \theta_{S_2}$ *where* $\omega_{S_1}, \omega_{S_2} \geq 0$, *we have* $\arg\max_j f_{\theta_{S_1 \cup S_2}}(x)[j] = y$.

For nonlinear classifiers, this intuition is almost correct according to our experiments, and Lemma 6.4 demonstrates the impact of the incorrectness on the estimated FSVs. Let $\hat{v}(\theta_S^t)$ denote the utility of aggregated model θ_S^t estimated under SampleSkip. Then, for all aggregated models θ_S^t, the incorrectness of SampleSkip can be measured by the percentage $\Delta v_S^t = \hat{v}(\theta_S^t) - v(\theta_S^t)$ of wrongly skipped test samples, which is upper bounded by Δv_{max}. Lemma 6.4 implies that Δv_{max} affects the error $|\hat{\phi}_i - \phi_i|$ of the estimated FSV $\hat{\phi}_i$, and clients can infer the upper bound of the error by estimating Δv_{max} with a small test set. Note that this upper bound is loose, and the exact error might be considerably smaller than it.

LEMMA 6.4. *For each client i, the error $|\hat{\phi}_i - \phi_i|$ of her FSV $\hat{\phi}_i$ estimated under SampleSkip is upper bounded by $T \cdot \Delta v_{max}$.*

6.1.3 Security. Ensuring that no private information can be learned from our system is impossible because the FSVs themselves are knowledge of clients' private models and test data. Conversely, we focus on some well-known attacks. A recent survey [40] suggests that the test phase of FL is threatened by model stealing and membership inference. Hence, we analyze the security of our systems against the membership inference attacks (simply *MI attacks* [19, 64]), the equation-solving model stealing attacks (simply *ES attacks* [71]), and the retraining-based model stealing attacks (simply *retraining attacks*, e.g., [26, 54, 55, 71]). The ES attacks can also be adapted for stealing test data (see Definition 6.5).

Definition 6.5 (Equation-solving attack). Let $f^{(l_1:l_2)}$ denote the inference function that given input features for layer l_1 yields output features for layer l_2, where $l_1 \leq l_2$. Given input features $X^{(l_1)}$ for layer l_1 and output features $\hat{Y}^{(l_2)}$ for layer l_2, an ES attack obtains the model parameters $\theta^{(l_1)}, ..., \theta^{(l_2)}$ of layers $l_1, ..., l_2$ by solving a system of equations $\hat{Y}^{(l_2)} = f^{(l_1:l_2)}(X^{(l_1)}; \{\theta^{(l)}\}_{l=l_1}^{l_2})$. We refer to it as a single-layer ES attack if $l_1 = l_2$; otherwise, as a multi-layer ES attack. Given model parameters $\theta^{(1)}, ..., \theta^{(l_2)}$ for layers $1, ..., l_2$ and output features $\hat{Y}^{(l_2)}$ for layer l_2, an ES data stealing attack learns the input features $X^{(1)}$ by solving $\hat{Y}^{(l_2)} = f^{(1:l_2)}(X^{(1)}; \{\theta^{(l)}\}_{l=1}^{l_2})$.

Definition 6.6 (Membership inference attack). Given a test sample (x, y) and its predicted label $\hat{y} = \arg\max_j f_\theta(x)[j]$, an MI attack determines whether (x, y) was in the training data of model θ.

Definition 6.7 (Retraining attack). Given the features X of some test samples and the predicted labels $\hat{Y} = \mathbf{Argmax}(f_\theta(X))$, a retraining attack learns a clone model f' such that $\mathbf{Argmax}(f'(X)) \approx \hat{Y}$.

Proposition 6.8. *Assume that linear layers are one-way functions, i.e., for all PPT adversaries $\mathcal{A}$ and for all linear layers l, there is a negligible ϵ such that $Pr[\mathcal{A}(\hat{Y}^{(l)}) = \theta^{(l)}] \leq \epsilon$ and $Pr[\mathcal{A}(\hat{Y}^{(l)}) = X^{(l)}] \leq \epsilon$ over all possible $(\theta^{(l)}, X^{(l)})$, where $\hat{Y}^{(l)} = lin^{(l)}(\theta^{(l)}, X^{(l)})$. Under SecSV or HESV, if $n \geq 4$, all PPT adversaries cannot apply single-layer ES attacks, ES data stealing attacks, MI attacks, or retraining attacks; if $n \geq L + 2$, they cannot apply multi-layer ES attacks.*

Proposition 6.8 characterizes different security levels of our protocols with different numbers of clients. Intuitively, to defend against single-layer ES attacks, for each layer l, HESV and SecSV should ensure that clients i_l, i_{l+1} are different entities such that they cannot know both the input $X^{(l)}$ and output $\hat{Y}^{(l)}$ to infer model parameters $\theta^{(l)}$. Similarly, clients $i_1, ..., i_{L+1}$ should not be the owner of the model under testing, which prevents ES data stealing attacks; client i_{L+1} who obtains the predicted labels $\hat{Y}$ should be different from client i_1 (the owner of the batch of samples) such that they cannot apply MI and retraining attacks. Note that if we have sufficient clients, SecSV can batch samples from different clients together; in this case, client i_1 represents the owners of the batched samples. For multi-layer ES attacks, HESV and SecSV require at least $L + 2$ clients to participate, which may be infeasible when L is large. However, because our clients are honest-but-curious, we can consider a lower security level where only 4 clients are required by ignoring the multi-layer ES attack whose success relies on a large number of random queries over the inference function $f^{(l_1:l_2)}$ by an active attacker; we can also slightly modify HESV and SecSV by applying garbled circuits to evaluate activation functions securely [27], which prevents the above attacks for any number of clients.

6.2 Discussion

6.2.1 Connection with secure federated training. The protocol for secure SV calculation is generic and independent of the protocol for secure federated training, as the local models are assumed to be given. Thus, the training method is out of the scope of this study. Additionally, owing to the use of the FSV, secure SV calculation can be inserted into secure federated training, giving the server flexibility to decide when to perform contribution evaluation. For instance, the server may promptly calculate SSVs after each training round to identify clients with negative contributions at an early stage; if the training task is urgent, the clients can prioritize finishing all the training rounds over secure SV calculation.

6.2.2 Parallelization. There are two basic strategies for parallelizing secure SV calculation. First, because the FSV is the sum of multiple rounds of SSVs, which are independent of each other, the server(s) can recruit a process for each round of SSV calculation to parallelize FSV calculation. Second, the server(s) can further parallelize each round of secure model testing by distributing the numerous models or test samples to multiple processes for evaluation. However, this strategy cannot be easily adopted for SampleSkip, under which the workload of testing a model may depend on the evaluation result of another model. Developing parallelization methods that are compatible with SampleSkip would be interesting.

6.2.3 Accelerating SV calculation. To calculate SSVs in a single round of FL, the server has to evaluate $O(2^n)$ models on M test samples, which requires testing models $O(2^n \cdot M)$ times. Therefore, we can reduce the scale of (1) models or (2) samples to be tested. The existing SV estimation methods [10, 13, 23, 43] use the first strategy: they sample and test only a subset of models for SV calculation. Skipping a portion of models does not introduce a significant estimation error because these methods assume that numerous clients participate in collaborative ML. However, in cross-silo FL, the number n of clients is relatively small, causing the failure of the above methods in skipping models. SampleSkip instead adopts the second strategy and is effective in those cases where model testing is a significant bottleneck, e.g., when HE is used. Moreover, we can use both strategies and easily combine SampleSkip with an existing model-skipping estimation method. Note that although n is small and our task is parallelizable, accelerating secure SV calculation is crucial because HE slows down SV calculation drastically.

6.2.4 Dynamic scenarios. Our methods also work when clients join in or leave FL halfway. For each client, we only need to calculate SSVs for the rounds she attended and add them up as her FSV.

6.2.5 Other variants of SV. Our techniques also support other FL-oriented variants of SV, including the Contribution Index [69], Group SV [42], GTG-Shapley [38], and WT-Shapley [78]. Intuitively, these metrics are functions of the utilities of local and aggregate models; thus, after securely testing model utilities by our methods, we can easily calculate them. This characterization also suggests that our methods could be adapted to other model utility-based contribution metrics for FL.

Table 2: Running time and error in detail. Brackets are used to indicate results under parallelization.

Dataset (model)	Protocol	Running time (s)						Speedup w.r.t. HESV (par.)	Slowdown w.r.t. NSSV (par.)	Error
		Total (par.)	Arithmetic	Enc.	Dec.	Comm.	Shares gen.			
AGNEWS (LOGI)	HESV	4400 (460)	3860	121	50.48	96.25	0.00	1× (1×)	28.3× (29.0×)	2.84×10^{-3}
	SecSV	1054 (126)	1005	6.61	1.77	8.53	7.84	4.2× (3.6×)	6.8× (8.0×)	9.64×10^{-4}
	SecretSV	3698 (376)	146	0.00	0.00	256	1713	1.2× (1.2×)	23.8× (23.7×)	2.46×10^{-3}
BANK (LOGI)	HESV	2934 (302)	2255	149	64.99	124	0.00	1× (1×)	13.5× (12.5×)	2.13×10^{-3}
	SecSV	137 (18.75)	113	1.08	1.14	4.06	7.81	21.4× (16.1×)	0.6× (0.8×)	8.86×10^{-4}
	SecretSV	791 (81.04)	26.64	0.00	0.00	55.17	376	3.7× (3.7×)	3.6× (3.3×)	1.00×10^{-3}
MNIST (CNN)	HESV	274470 (27600)	76384	87610	290	29658	0.00	1× (1×)	276.9× (274.2×)	6.98×10^{-4}
	SecSV	39310 (3992)	32468	11.27	199	592	2299	7.0× (6.9×)	39.7× (39.7×)	9.28×10^{-4}
	SecretSV	158407 (16003)	3751	0.00	0.00	8933	121393	1.7× (1.7×)	159.8× (159.0×)	1.20×10^{-3}
miRNA-mRNA (RNN)	HESV	411628 (41691)	190190	61528	738	21481	0.00	1× (1×)	47.9× (48.4×)	1.40×10^{-2}
	SecSV	77081 (7825)	30215	656	321	1288	3050	5.3× (5.3×)	9.0× (9.1×)	1.70×10^{-2}
	SecretSV	38567 (3902)	1935	0.00	0.00	2796	22346	10.7× (10.7×)	4.5× (4.5×)	1.87×10^{-0}

Table 3: Comparisons of SV estimation methods under SecSV.

Dataset (model)	Method	Speedup w.r.t. HESV		Error ($\times 10^{-2}$)	
		SampleSkip off/on		SampleSkip off/on	
AGNEWS (LOGI)	SecSV	4.2 ×	7.2×	0.10	0.10
	SecSV+PS	4.2 ×	7.2×	2.00	2.01
	SecSV+GT	3.5 ×	5.5×	3.41	3.39
	SecSV+KS	5.3 ×	8.6×	17.63	17.63
BANK (LOGI)	SecSV	21.4 ×	36.6×	0.09	0.09
	SecSV+PS	21.3 ×	36.5×	1.25	1.24
	SecSV+GT	8.9 ×	10.8×	3.40	3.40
	SecSV+KS	27.0 ×	44.1×	7.67	7.66
MNIST (CNN)	SecSV	7.0 ×	25.8×	0.09	0.64
	SecSV+PS	7.0 ×	25.8×	2.69	2.88
	SecSV+GT	6.9 ×	25.3×	3.58	3.80
	SecSV+KS	9.0 ×	27.2×	15.46	15.65
miRNA-mRNA (RNN)	SecSV	5.3 ×	11.8×	1.70	1.82
	SecSV+PS	5.3 ×	11.8×	3.03	3.25
	SecSV+GT	5.3 ×	11.7×	3.67	3.50
	SecSV+KS	7.0 ×	14.0×	20.77	20.49

Table 4: Speedup of Matrix Reducing w.r.t. Matrix Squaring in the time per sample spent on HE computations for evaluating *AB*. The shape of matrix *A* is varied. "Full" means both *A* and *B* are encrypted, whilst "Half" means only *A* is encrypted.

Shape	4×300	2×48	64×256	10×64	32×64	32×32	2×32
Full	1.69×	6.10×	1.99×	2.30×	2.66×	2.85×	2.45×
Half	3.24×	11.39×	3.92×	4.49×	5.23×	3.71×	2.87×

7 EXPERIMENTS

7.1 Setup

Research questions. We experiment to answer these questions.

- *Evaluation efficiency (RQ1)*: How efficient are SecSV and HESV for secure SV calculation? How much can SampleSkip accelerate SecSV? How efficient are Matrix Squaring and Matrix Reducing for secure matrix multiplication?
- *Evaluation accuracy (RQ2)*: How accurately do SecSV and HESV calculate FSVs? How many test samples are wrongly identified as skippable samples by SampleSkip?
- *Parameters' effects (RQ3)*: What are the effects of the size M of test samples, number n of clients, and number L of layers on evaluation efficiency and accuracy?

Baselines. For secure SV calculation, we compare SecSV with HESV and Nonsecure SV (NSSV), which calculates exact FSVs in a nonsecure environment. In addition, we design a two-server baseline protocol named *SecretSV* for comparison, which protects models and test data purely by ASS. See Appendix A to check the details of NSSV and SecretSV. For SV estimation, we compare SampleSkip with three widely adopted methods: *Permutation Samples (PS)* [43], *Group Testing (GT)* [23], and *KernelSHAP (KS)* [39].

Datasets and classifiers. We use the *AGNEWS* [85], *BANK* [50], *MNIST* [31], and *miRNA-mRNA* [47] datasets, which cover the news classification, bank marketing, image recognition, and miRNA target prediction tasks, for experiments. By default, we consider logistic classifiers (LOGI) for AGNEWS and BANK, a convolutional NN (CNN) [24] for MNIST, and a recurrent NN (RNN) [32] for miRNA-mRNA; when evaluating the effect of the number L of layers, we rather focus on the deep NN (DNN) to easily vary L, which consists of an input layer, an output layer, and $L - 2$ hidden layers. See Appendix C for details of the datasets and classifiers.

Federated training. We allocate each label of training and test samples to clients according to the Dirichlet distribution of order n with scaling parameters of 0.5 to construct non-IID local data, following [81]. Subsequently, $n = 5$ clients perform $T = 10$ rounds of the FedAvg algorithm [44] to train the above classifiers.

Evaluation metrics. The computation efficiency of a secure SV calculation protocol is measured by its speedup w.r.t. the baseline HESV. Subsequently, following [23], we use the Euclidean distance to measure the error of the estimated FSVs. Concretely, let $\hat{\boldsymbol{\phi}}$ and $\boldsymbol{\phi}$ denote the vectors of the estimated FSVs and the exact FSVs for all the clients. The (expected) error of $\hat{\boldsymbol{\phi}}$ is defined as $\mathbb{E}_{\boldsymbol{\phi}}[||\hat{\boldsymbol{\phi}} - \boldsymbol{\phi}||_2]$. Note that both HE and ASS introduce noise into messages, and the exact FSVs are calculated by the nonsecure protocol NSSV.

Environment. We implement the server and clients on a workstation with a 3.0 GHz 16-core processor and 128 GB RAM and logically simulate 1 Gbps communication channels between the parties. When

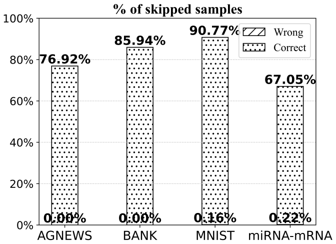


Figure 5: Skipped samples.

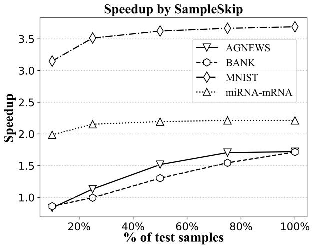


Figure 6: Effect of M.

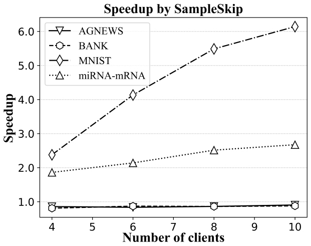


Figure 7: Effect of n.

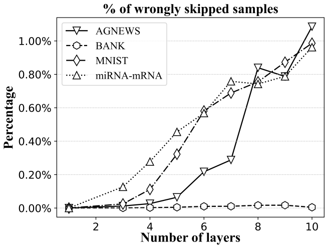


Figure 8: Effect of L.

evaluating efficiency under parallelization, we allocate a dedicated CPU core for each round of SSV calculation. We use TenSEAL [3], a Python wrapper for Microsoft's SEAL library, to implement HE operations and select the parameters of HE and ASS for 128-bit security based on the *Homomorphic Encryption Security Standard* [1], which results in $N = 2048$ ciphertext slots. We run each experiment 10 times and present the average results.

7.2 Experimental Results

7.2.1 Evaluation efficiency (RQ1). Table 2 shows the running time of different protocols spent on secure SV calculation. We can see that HESV requires a considerably long running time even under parallelization and is dramatically less efficient than NSSV because the use of HE is extremely expensive in computation. This verifies the importance of SV estimation in our setting. The slowdown effect could be more significant if we run NSSV on GPUs. SecSV spends considerably less time than HESV on HE computations owing to the avoidance of c2c multiplications; the time it takes to generate secret shares is insignificant compared with the total running time. Although SecSV requires communication with an auxiliary server, it may consume less communication time than HESV because the size of a message's secret shares is significantly smaller than that of its ciphertext. SecretSV performs arithmetic operations efficiently as models and test data are not encrypted. However, shares generation and communication are time-consuming because a triplet of random masks should be secretly shared between the two servers for every multiplication under ASS.

Table 3 compares SampleSkip with the existing SV estimation methods in terms of acceleration. Whereas SampleSkip and KS significantly speed up SecSV, the PS and GT methods hardly accelerate it because they fail to skip any models. For the logistic classifiers, GT significantly slows down SecSV because it needs to calculate FSVs by solving an optimization problem, which is expensive for light classifiers. Furthermore, SampleSkip can be combined with all these baselines to significantly improve efficiency. This marvelous performance is caused by the high percentage of test samples skipped by SampleSkip, as shown in Figure 5.

Finally, we evaluate the efficiency of Matrix Squaring and Matrix Reducing. As seen in Table 4, we evaluate secure matrix multiplications between a $d_{out} \times d_{in}$ matrix A and a $d_{in} \times m$ matrix B; we set batch size $m = \lfloor N/d_{out} \rfloor$ for Matrix Reducing and $m = \min\{d_{in}, \lfloor\sqrt{N}\rfloor\}$ for Matrix Squaring. Subsequently, we enumerate all the matrix multiplications involved in the four classifiers considered in our experiments and present the speedup of Matrix Reducing w.r.t. Matrix Squaring in each case. When numerous homomorphic rotations are required for Matrix Squaring, e.g., when A is of shapes 2×48, the computation time per sample under Matrix Reducing is considerably less than that under Matrix Squaring. Additionally, compared with the case where both A and B are encrypted, when only A is encrypted, the speedup of Matrix Reducing is higher because the computation time required for homomorphic multiplications becomes less whereas that required for homomorphic rotations remains the same.

7.2.2 Evaluation accuracy (RQ2). As shown in Table 2, both SecSV and HESV can accurately calculate FSVs for all datasets. The error of the FSVs calculated by SecretSV is passable for the logistic and CNN classifiers but overly large for RNN. This may be because RNN has numerous linear layers in sequence, through which ASS introduces accumulated truncation errors into prediction results.

Table 3 suggests that SampleSkip introduces only an insignificant error for SecSV even when combined with other estimation methods. The PS and GT methods vastly increase the error because they approximate the FSVs even if they skip no models. The KS method skips some models and accelerates SecSV to some extent but also results in an unacceptable error for all the datasets because it works poorly when the number n of clients is small.

We also present the percentages of the samples skipped by SampleSkip in Figure 5. It is evident that no samples are wrongly skipped for the linear logistic classifiers, which experimentally demonstrates the correctness of Theorem 6.3. For the nonlinear CNN and RNN classifiers, only a tiny portion of samples are wrongly skipped, which results in a minor error.

7.2.3 Parameters' effects (RQ3). First, we vary the size M of test samples to evaluate its effect on the efficiency of SecSV. As depicted in Figure 6, when more test samples are used for evaluation, SampleSkip becomes more effective in accelerating SecSV. This effect may be due to two reasons. First, SecSV usually requires a large batch size to perform at its best owing to the use of Matrix Reducing. When the size is small, and the majority of the samples are skipped, the remaining discriminative samples might be overly few to fulfill the batch, which causes a waste of ciphertext slots. Second, SampleSkip needs to preprocess discriminative samples for each aggregated model, which is relatively time-consuming, particularly for light classifiers. Thus, for the BANK and AGNEWS datasets with logistic models, when only a few samples are evaluated, SampleSkip may slow down SecSV. Note that we can easily avoid this problem by adaptively turning off SampleSkip.

We then vary the number n of clients to test its effect on the efficiency of SecSV. In this case, we use 10% of the entire test samples. In Figure 7, we can see that when n increases, SampleSkip accelerates SecSV more for MNIST and miRNA-mRNA but probably less for AGNEWS and BANK. The reason for this effect is as follows. An increase of n implies a higher percentage of aggregated models w.r.t. all models because the number of aggregated models increases exponentially w.r.t. n. The increase in the percentage of aggregated models can amplify the speedup and slowdown effects of SampleSkip. Therefore, because SampleSkip speeds up SecSV for MNIST and miRNA-mRNA in Figure 6, a larger n results in a higher speedup. However, because SampleSkip slows down SecSV when only 10% of the test samples are used owing to the need to preprocess discriminative samples, a larger n may cause a higher slowdown. Note that the lines for AGNEWS and BANK in Figure 7 may rise because more models need to be securely tested; this makes the speedup approach 1.0.

Finally, we vary the number L of layers in the DNN classifier. Figure 8 shows that when $L = 1$, which means DNN is reduced to a logistic classifier because no hidden layer exists, no samples are wrongly skipped, which verifies Theorem 6.3. When L increases, DNN becomes "more nonlinear"; thus, more samples may be wrongly skipped. However, the percentages of the wrongly skipped samples remain small even when $L = 10$.

8 RELATED WORK

Secure FL. There are two typical FL settings: cross-device and cross-silo [30]. In the cross-device setting, the clients are numerous unreliable mobile or IoT devices with limited computation and communication resources, making using lightweight cryptographic techniques to protect clients' privacy reasonable. Therefore, existing secure cross-device FL systems [2, 4, 6, 16, 28, 66–68] usually protect privacy using secure aggregation: they mask the clients' updates using secret sharing, and the mask for each update will be canceled by aggregating the masked updates. Nevertheless, secure aggregation can only protect the local updates; the final model is still leaked to the FL server or a third party. Some studies [12, 15, 20, 29, 35, 45, 52, 72, 87] protect privacy against a third party using differential privacy (DP) [7] or even against the server using local DP [8]. However, the uses of DP and local DP significantly compromise the final model's utility. Conversely, the clients in cross-silo FL are a small number of organizations with abundant communication and computing resources. Considering that the purpose of the clients is to train an accurate model for their own use, they might not allow releasing the final model to external parties, including the server, and/or tolerate a nontrivial loss of model utility [82]. Consequently, HE is a natural choice for cross-silo FL, which is advocated by many works [25, 41, 57, 60, 73, 82, 84]. Although HE largely increases the computation and communication overhead, it is acceptable to those powerful clients. However, the existing HE-based FL systems cannot support secure SV calculation because they only protect the model aggregation process, which inspires this paper. Additionally, Ma et al. [42] studied a similar problem to ours; however, they aimed to make the contribution evaluation process transparent and auditable using blockchain, whereas we focus on the security of this process.

SV for collaborative ML. The SV has been widely adopted in collaborative ML for contribution evaluation and numerous downstream tasks. Concretely, because the value of data is task-specific, the contribution metric SV can be considered a data valuation metric [13, 23, 76, 77]. Some studies [17, 36, 53, 69] proposed SV-based payment schemes that allocate monetary payments to clients based on their respective SVs. The evaluated SVs can also be used to decide the qualities of models [65] or synthetic data [70] that are rewarded to clients and to identify irrelevant clients who may have negative contributions to the final model [51].

Given that SV calculation requires exponential times of model retraining and evaluation to accelerate SV calculation, various estimation methods [10, 13, 23, 38, 43, 69, 76, 77] were proposed, and they can be categorized into two: *retraining-skipping* and *model-skipping*. The former class of methods reduces the complexity of or even eliminates the redundant retraining process. For example, the FSV [76], MR [69], Truncated MR [77], and GTG-Shapley [38] methods use the local and aggregated models for SV calculation, which avoids retraining FL models. The model-skipping methods, e.g., the PS [43] and GT [23] methods used in our experiments, approximate the SV by sampling a sufficient number of models for evaluation. Our SampleSkip method skips test samples to accelerate the model evaluation process, which does not belong to the above classes and opens up a new direction: *sample-skipping estimation.*

Secure inference. This problem regards securely making predictions given a client's encrypted input features for prediction-as-a-service (PaaS). The majority of existing studies (e.g., [14, 27, 46, 48, 58]) assumed that the prediction model is not outsourced because the server of PaaS is usually the model owner himself and thus do not encrypt the model, which is different from our privacy model. Several studies [18, 24, 49] examined the scenario of *secure outsourced inference* where the model is outsourced to and should be protected against an untrusted server, which is similar to our scenario. Nevertheless, in our problem, we need to ensure security against more untrusted parties, and the server(s) should securely evaluate numerous models on massive test samples, which calls for estimation methods to reduce the scale of the task.

9 CONCLUSION

In this paper, we introduced the problem of secure SV calculation for cross-silo FL and proposed two protocols. The one-server protocol HESV had a stronger security guarantee because it only required a single server to coordinate secure SV calculation, whereas the two-server protocol SecSV was considerably more efficient. Solving this problem facilitates many downstream tasks and is an essential step toward building a trustworthy ecosystem for cross-silo FL. Future directions include studying secure SV calculation in the vertical FL setting, where clients possess different attributes of the same data samples, and developing secure protocols for other types of prediction models and more effective acceleration techniques.

ACKNOWLEDGMENTS

We are grateful for the generous support provided by JSPS to the first author, a JSPS Research Fellow, as well as the support from JST CREST (No. JPMJCR21M2), JST SICORP (No. JPMJSC2107), and JSPS KAKENHI (No. 17H06099, 21J23090, 21K19767, 22H03595).

REFERENCES

[1] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2018. *Homomorphic Encryption Security Standard*. Technical Report. HomomorphicEncryption.org.

[2] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1253–1269.

[3] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. 2021. TenSEAL: A library for encrypted tensor operations using homomorphic encryption. *arXiv preprint arXiv:2104.03152* (2021).

[4] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1175–1191.

[5] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017*. 409–437.

[6] Beongjun Choi, Jy-yong Sohn, Dong-Jun Han, and Jaekyun Moon. 2020. Communication-computation efficient secure aggregation for federated learning. *arXiv preprint arXiv:2012.05433* (2020).

[7] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*. 265–284.

[8] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. 2003. Limiting privacy breaches in privacy preserving data mining. In *ACM SIGMOD*. 211–222.

[9] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. 2020. Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948* (2020).

[10] Shaheen S. Fatima, Michael Wooldridge, and Nicholas R. Jennings. 2008. A linear approximation method for the Shapley value. *Artificial Intelligence* 172, 14 (2008), 1673–1699.

[11] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in Neural Information Processing Systems* 33 (2020), 16937–16947.

[12] Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).

[13] Amirata Ghorbani and James Zou. 2019. Data shapley: Equitable valuation of data for machine learning. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. 2242–2251.

[14] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*. 201–210.

[15] Antonious M Girgis, Deepesh Data, Suhas Diggavi, Peter Kairouz, and Ananda Theertha Suresh. 2021. Shuffled model of federated learning: Privacy, accuracy and communication trade-offs. *IEEE journal on selected areas in information theory* 2, 1 (2021), 464–478.

[16] Xiaojie Guo, Zheli Liu, Jin Li, Jiqiang Gao, Boyu Hou, Changyu Dong, and Thar Baker. 2020. V eri fl: Communication-efficient and fast verifiable aggregation for federated learning. *IEEE Transactions on Information Forensics and Security* 16 (2020), 1736–1751.

[17] Dongge Han, Michael Wooldridge, Alex Rogers, Shruti Tople, Olga Ohrimenko, and Sebastian Tschiatschek. 2020. Replication-robust payoff-allocation for machine learning data markets. *arXiv preprint arXiv:2006.14583* (2020).

[18] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N Wright. 2018. Privacy-preserving Machine Learning as a Service. *Proc. Priv. Enhancing Technol.* 2018, 3 (2018), 123–142.

[19] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S Yu, and Xuyun Zhang. 2022. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)* 54, 11s (2022), 1–37.

[20] Rui Hu, Yuanxiong Guo, Hongning Li, Qingqi Pei, and Yanmin Gong. 2020. Personalized federated learning with differential privacy. *IEEE Internet of Things Journal* 7, 10 (2020), 9530–9539.

[21] Yutao Huang, Lingyang Chu, Zirui Zhou, Lanjun Wang, Jiangchuan Liu, Jian Pei, and Yong Zhang. 2021. Personalized cross-silo federated learning on non-iid data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 7865–7873.

[22] Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nezihe Merve Gurel, Bo Li4 Ce Zhang, and Costas Spanos1 Dawn Song. 2019. Efficient Task-Specific Data Valuation for Nearest Neighbor Algorithms. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1610–1623.

[23] Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nick Hynes, Nezihe Merve Gürel, Bo Li, Ce Zhang, Dawn Song, and Costas J Spanos. 2019. Towards efficient data valuation based on the shapley value. In *The 22nd International Conference on Artificial Intelligence and Statistics*. 1167–1176.

[24] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure Outsourced Matrix Computation and Application to Neural Networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1209–1222.

[25] Zhifeng Jiang, Wei Wang, and Yang Liu. 2021. FLASHE: Additively symmetric homomorphic encryption for cross-silo federated learning. *arXiv preprint arXiv:2109.00675* (2021).

[26] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. 2019. PRADA: protecting against DNN model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. 512–527.

[27] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*. 1651–1669.

[28] Swanand Kadhe, Nived Rajaraman, O Ozan Koyluoglu, and Kannan Ramchandran. 2020. FASTSECAGG: Scalable secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2009.11248* (2020).

[29] Peter Kairouz, Ziyu Liu, and Thomas Steinke. 2021. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *International Conference on Machine Learning*. 5201–5212.

[30] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* 14, 1–2 (2021), 1–210.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[32] Byunghan Lee, Junghwan Baek, Seunghyun Park, and Sungroh Yoon. 2016. deepTarget: End-to-end learning framework for microRNA target prediction using deep recurrent neural networks. In *Proceedings of the 7th ACM international conference on bioinformatics, computational biology, and health informatics*. 434–442.

[33] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.

[34] Jinfei Liu, Jian Lou, Junxu Liu, Li Xiong, Jian Pei, and Jimeng Sun. 2021. Dealer: An end-to-end model marketplace with differential privacy. *Proceedings of the VLDB Endowment* 14, 6 (2021), 957–969.

[35] Ruixuan Liu, Yang Cao, Hong Chen, Ruoyang Guo, and Masatoshi Yoshikawa. 2021. FLAME: Differentially private federated learning in the shuffle model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8688–8696.

[36] Yuan Liu, Zhengpeng Ai, Shuai Sun, Shuangfeng Zhang, Zelei Liu, and Han Yu. 2020. *FedCoin: A Peer-to-Peer Payment System for Federated Learning*. Springer International Publishing, 125–138.

[37] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. 2020. A secure federated transfer learning framework. *IEEE Intelligent Systems* 35, 4 (2020), 70–82.

[38] Zelei Liu, Yuanyuan Chen, Han Yu, Yang Liu, and Lizhen Cui. 2022. GTG-Shapley: Efficient and Accurate Participant Contribution Evaluation in Federated Learning. *ACM Transactions on Intelligent Systems and Technology (TIST)* 13, 4 (2022), 1–21.

[39] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems*, Vol. 30.

[40] Lingjuan Lyu, Han Yu, Xingjun Ma, Chen Chen, Lichao Sun, Jun Zhao, Qiang Yang, and Philip S. Yu. 2022. Privacy and Robustness in Federated Learning: Attacks and Defenses. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[41] Jing Ma, Si-Ahmed Naas, Stephan Sigg, and Xixiang Lyu. 2022. Privacy-preserving federated learning based on multi-key homomorphic encryption. *International Journal of Intelligent Systems* (2022).

[42] Shuaicheng Ma, Yang Cao, and Li Xiong. 2021. Transparent contribution evaluation for secure federated learning on blockchain. In *2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 88–91.

[43] Sasan Maleki. 2015. *Addressing the computational issues of the Shapley value with applications in the smart grid*. Ph.D. Dissertation. University of Southampton.

[44] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. 1273–1282.

[45] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2017. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963* (2017).

[46] Souhail Meftah, Benjamin Hong Meng Tan, Chan Fook Mun, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Vijay Chandrasekhar. 2021. DOReN: Toward Efficient Deep Convolutional Neural Networks with Fully Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security* 16 (2021), 3740–3752.

[47] Mark Menor, Travers Ching, Xun Zhu, David Garmire, and Lana X Garmire. 2014. mirMark: a site-level and UTR-level classifier for miRNA target prediction.

Genome biology 15, 10 (2014), 1–16.

[48] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium (USENIX Security 20)*. 2505–2522.

[49] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 19–38.

[50] Sérgio Moro, Paulo Cortez, and Paulo Rita. 2014. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems* 62 (2014), 22–31.

[51] Lokesh Nagalapatti and Ramasuri Narayanam. 2021. Game of Gradients: Mitigating Irrelevant Clients in Federated Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 10 (2021), 9046–9054.

[52] Maxence Noble, Aurélien Bellet, and Aymeric Dieuleveut. 2022. Differentially private federated learning on heterogeneous data. In *International Conference on Artificial Intelligence and Statistics*. 10110–10145.

[53] Olga Ohrimenko, Shruti Tople, and Sebastian Tschiatschek. 2019. Collaborative machine learning markets with data-replication-robust payments. *arXiv preprint arXiv:1911.09052* (2019).

[54] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. 2019. Knockoff Nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 4954–4963.

[55] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 506–519.

[56] Matthias Paulik et al. 2021. Federated evaluation and tuning for on-device personalization: System design & applications. *arXiv:2102.08503* (2021).

[57] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. 2018. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security* 13, 5 (2018), 1333–1345.

[58] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T. Lee, Hsien-Hsin S. Lee, Gu-Yeon Wei, and David Brooks. 2021. Cheetah: Optimizing and Accelerating Homomorphic Encryption for Private Inference. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 26–39.

[59] M Sa degh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 707–721.

[60] Sinem Sav, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. 2021. POSEIDON: Privacy-Preserving Federated Neural Network Learning. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*.

[61] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (nov 1979), 612–613.

[62] L. S. Shapley. 1953. *A Value for n-Person Games*. Princeton University Press, 307–318.

[63] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 1310–1321.

[64] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 3–18.

[65] Rachael Hwee Ling Sim, Yehong Zhang, Mun Choon Chan, and Bryan Kian Hsiang Low. 2020. Collaborative Machine Learning with Incentive-Aware Model Rewards. In *Proceedings of the 37th International Conference on Machine Learning*. 8927–8936.

[66] Jinhyun So, Başak Güler, and A Salman Avestimehr. 2020. Byzantine-resilient secure federated learning. *IEEE Journal on Selected Areas in Communications* 39, 7 (2020), 2168–2181.

[67] Jinhyun So, Başak Güler, and A Salman Avestimehr. 2021. Turbo-Aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory* 2, 1 (2021), 479–489.

[68] Jinhyun So, Corey J Nolet, Chien-Sheng Yang, Songze Li, Qian Yu, Ramy E Ali, Basak Guler, and Salman Avestimehr. 2022. LightSecAgg: a lightweight and versatile design for secure aggregation in federated learning. *Proceedings of Machine Learning and Systems* 4 (2022), 694–720.

[69] Tianshu Song, Yongxin Tong, and Shuyue Wei. 2019. Profit Allocation for Federated Learning. In *2019 IEEE International Conference on Big Data (Big Data)*. 2577–2586.

[70] Sebastian Shenghong Tay, Xinyi Xu, Chuan Sheng Foo, and Bryan Kian Hsiang Low. 2022. Incentivizing collaboration in machine learning via synthetic data rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9448–9456.

[71] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*. 601–618.

[72] Aleksei Triastcyn and Boi Faltings. 2019. Federated learning with bayesian differential privacy. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2587–2596.

[73] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. 2019. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM workshop on artificial intelligence and security*. 1–11.

[74] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.* 2019, 3 (2019), 26–49.

[75] Kangkang Wang et al. 2019. Federated evaluation of on-device personalization. *arXiv:1910.10252* (2019).

[76] Tianhao Wang, Johannes Rausch, Ce Zhang, Ruoxi Jia, and Dawn Song. 2020. A principled approach to data valuation for federated learning. In *Federated Learning*. Springer, 153–167.

[77] Shuyue Wei, Yongxin Tong, Zimu Zhou, and Tianshu Song. 2020. *Efficient and Fair Data Valuation for Horizontal Federated Learning*. Springer International Publishing, 139–152.

[78] Chengyi Yang, Jia Liu, Hao Sun, Tongzhi Li, and Zengxiang Li. 2022. WTDP-Shapley: Efficient and Effective Incentive Mechanism in Federated Learning for Intelligent Safety Inspection. *IEEE Transactions on Big Data* (2022), 1–10.

[79] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.

[80] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. 2021. See through gradients: Image batch recovery via gradinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16337–16346.

[81] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. 2019. Bayesian Nonparametric Federated Learning of Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*. 7252–7261.

[82] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. 2020. BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 493–506.

[83] Qiao Zhang, Cong Wang, Hongyi Wu, Chunsheng Xin, and Tran V. Phuong. 2018. GELU-Net: A Globally Encrypted, Locally Unencrypted Deep Neural Network for Privacy-Preserved Learning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 3933–3939.

[84] Shulai Zhang, Zirui Li, Quan Chen, Wenli Zheng, Jingwen Leng, and Minyi Guo. 2021. Dubhe: Towards data unbiasedness with homomorphic encryption in federated learning client selection. In *50th International Conference on Parallel Processing*. 1–10.

[85] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems* 28 (2015).

[86] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2020. iDLG: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610* (2020).

[87] Yang Zhao, Jun Zhao, Mengmeng Yang, Teng Wang, Ning Wang, Lingjuan Lyu, Dusit Niyato, and Kwok-Yan Lam. 2020. Local differential privacy-based federated learning for internet of things. *IEEE Internet of Things Journal* 8, 11 (2020), 8836–8853.

[88] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep Leakage from Gradients. In *Advances in Neural Information Processing Systems*. 14747–14756.

ALGORITHM 8: Nonsecure SV (NSSV)

```
Each client $i$: Submit test data $D_i$ to Server
for $t$ in $\{1, ..., T\}$ do
    for $S \subseteq I^t$, $|S| > 0$ do
        Server: Calculate $\theta_S^t$ and evaluate its utility $v(\theta_S^t)$
    Server $\mathcal{P}$: Calculate SSVs $\phi_1^t, ..., \phi_n^t$
Server $\mathcal{P}$: Calculate FSVs $\phi_1, ..., \phi_n$
return $\phi_1, ..., \phi_n$
```

ALGORITHM 9: SecretSV

```
Server $\mathcal{P}$: Randomly select a leader client
Leader: Generate a public key $pk$ and a private key $sk$ of HE and
    broadcast them among the other clients
Each client $i$: Generate secret shares $\theta_i'^t, \theta_i''^t$ of $\theta_i^t$, send $\theta_i'^t$ to server
    $\mathcal{P}$, and send $\theta_i''^t$ to server $\mathcal{A}$.
Each client $i$: Generate secret shares $D_i' = (X_i', Y_i'), D_i'' = (X_i'', Y_i'')$ of
    $D_i = (X_i, Y_i)$, send $D_i'$ to server $\mathcal{P}$, and send $D_i''$ to server $\mathcal{A}$
for $t$ in $\{1, ..., T\}$ do
    for $S \subseteq I^t$, $|S| > 0$ do
        Servers $\mathcal{P}, \mathcal{A}$: Calculate $\theta_S'^t$ and $\theta_S''^t$
        Server $\mathcal{P}$: Run Algorithm 10 $\Pi_{ASS}((\theta_S'^t, \theta_S''^t), (\mathcal{D}', \mathcal{D}''))$ to
            obtain $cnt_S^t$
        Server $\mathcal{P}$: Calculate $v(\theta_S^t) = cnt_S^t / M$
Server $\mathcal{P}$: Calculate SSVs $\phi_1^t, ..., \phi_n^t, \forall t \in [T]$
Server $\mathcal{P}$: Calculate FSVs $\phi_1, ..., \phi_n$
return $\phi_1, ..., \phi_n$
```

ALGORITHM 10: Π_{ASS}: Secure Testing for SecretSV

```
Input: secret shares $\theta'^t, \theta''^t$ of model, and secret shares $\mathcal{D}', \mathcal{D}'$ of
    collective test set
Output: count $cnt$ of correct predictions
$cnt \leftarrow 0$
for each $D' = (X', Y'), D'' = (X'', Y'') \in (\mathcal{D}', \mathcal{D}'')$ do
    for each model layer $l \in \{1, ..., L\}$ do
        Server $\mathcal{A}$: Calculate $\hat{Y}''^{(l)} = lin^{(l)}(\theta''^t, X''^{(l)})$ and send $\hat{Y}''^{(l)}$
            to client $i_{l+1} \in I \backslash \{i_l\}$, where $X''^{(1)} = X''$
        Server $\mathcal{P}$: Calculate $\hat{Y}'^{(l)} = lin^{(l)}(\theta'^t, X'^{(l)})$ and send $\hat{Y}'^{(l)}$ to
            client $i_{l+1} \in I \backslash \{i_l\}$, where $X'^{(1)} = X'$
        if $l < L$ then
            Client $i_{l+1}$: Compute $X^{(l+1)} = ac^{(l)}((\hat{Y}'^{(l)} + \hat{Y}''^{(l)}) \bmod p)$,
                generate shares $X'^{(l+1)}$ and $X''^{(l+1)}$, and send them to servers
                $\mathcal{P}$ and $C$, respectively
        else
            Client $i_{L+1}$: Calculate $\hat{Y} = \mathbf{Argmax}((\hat{Y}'^{(L)} + \hat{Y}''^{(L)}) \bmod p)$,
                generate shares $\hat{Y}', \hat{Y}''$, and send them to servers $\mathcal{P}$ and $C$,
                respectively
    Server $\mathcal{A}$: Compute and send $\tilde{Y}'' = \hat{Y}'' - Y''$ to server $\mathcal{P}$.
    Server $\mathcal{P}$: Calculate $\tilde{Y} = abs((\hat{Y}' - Y' + \tilde{Y}'') \bmod p)$, where $abs$
        denotes taking the entrywise absolute value.
    Server $\mathcal{P}$: $cnt \leftarrow cnt + \sum_{k=1}^{m} \mathbf{1}(|\tilde{Y}[k]| < 0.5)$
return $cnt$
```

A MISSING PROTOCOLS

The Nonsecure SV (NSSV) and SecretSV protocols are presented in Algorithms 8 and 9. NSSV just calculates the ground-truth FSVs in a nonsecure environment. SecretSV protects models and test data based purely on ASS, and its secure testing protocol (Algorithm 10) is adapted from a SOTA three-party ASS-based secure ML protocol [74]. The method for matrix multiplications between secret shares can be found in [74].

B MISSING PROOFS

Proof of Lemma 5.2. For all $j \in [d_{out}]$ and $k \in [m]$, we have

$$\begin{aligned}
\delta(A,B)[j,k] &= \sum_{o=1}^{d_{in}} \xi^{(o-1)}(\sigma(A))[j,k] \cdot \psi^{(o-1)}(\tau(B))[j,k] \\
&= \sum_{o=1}^{d_{in}} \sigma(A)[j,k+o-1] \cdot \tau(B)[j+o-1,k] \\
&= \sum_{o=1}^{d_{in}} A[j,j+k+o-1] \cdot B[j+k+o-1,k] \\
&= \sum_{o=1}^{d_{in}} A[j,o] \cdot B[o,k] = (AB)[j,k]
\end{aligned}$$

□

Lemma B.1. *To evaluate AB under HE with $d_{out} \le d_{in}$ and $m \le \min\{d_{in}, \lfloor\sqrt{N}\rfloor\}$, Matrix Squaring needs $O(\frac{d_{in} \cdot d_{out}}{\sqrt{N}})$ HMults and $O(\frac{d_{in}}{d_{out} \bmod \sqrt{N}})$ HRots.*

Proof of Lemma 5.3. Let $A' = (A \,|\, A)$, $B' = (B_1 \,|\, B_2)$, and $B'' = (B'; B')$. By running the first two steps of $\delta(A', B')$ and $\delta(A', B'')$, we derive two sets of matrices $\{\tilde{A}'^{(o)}\}_{o=1}^{d_{in}}$ and $\{\tilde{B}'^{(o)}\}_{o=1}^{d_{in}}$ for $\delta(A', B')$ and $\{\tilde{A}'^{(o)}\}_{o=1}^{2d_{in}}$ and $\{\tilde{B}''^{(o)}\}_{o=1}^{2d_{in}}$ for $\delta(A', B'')$, respectively, where $\forall o \in [2d_{in}], \tilde{A}'^{(o)} = \tilde{A}'^{((o-1) \bmod d_{in}+1)}, \tilde{B}''^{(o)} = \tilde{B}'^{((o-1) \bmod d_{in}+1)}$ Obviously, we have $2\delta(A', B') = \delta(A', B'')$ and can conclude that

$$\begin{aligned}
&(\delta(A,B_1) \,|\, \delta(A,B_2)) = (AB_1 \,|\, AB_2) = AB' \\
=&\frac{1}{2}A'B'' = \frac{1}{2}\delta(A', B'') = \delta(A', B') = \delta((A \,|\, A), (B_1 \,|\, B_2))
\end{aligned}$$

□

Proof of Lemma B.1. When $d_{out} \le d_{in} \le \lfloor\sqrt{N}\rfloor$, Matrix Squaring, i.e., the SOTA method, pads A with zeros to derive a $d'_{out} \times d_{in}$ matrix A' such that d'_{out} exactly divides d_{in}. In the best case where d_{out} exactly divides d_{in}, we have $d'_{out} = d_{out}$, while in the worst case where no number $d \in (d_{out}, d_{in})$ that exactly divides d_{in} exists, we have $d'_{out} = d_{in}$. Then, it needs d'_{out} homomorphic multiplications and $\lfloor\log(d_{in}/d'_{out})\rfloor + d_{in}/d'_{out} - 2^{\lfloor\log(d_{in}/d'_{out})\rfloor}$ homomorphic rotations to evaluate AB, which corresponds to $O(d_{in})$ homomorphic multiplications and $O(d_{in}/d_{out})$ homomorphic rotations.

When $d_{in} > \lfloor\sqrt{N}\rfloor \ge d_{out}$, Matrix Squaring, i.e., the SOTA method with our extension, splits A and B to derive $\lceil d_{in}/\lfloor\sqrt{N}\rfloor\rceil$ pairs of $d_{out} \times \lfloor\sqrt{N}\rfloor$ matrices and $\lfloor\sqrt{N}\rfloor \times m$ matrices and applies the SOTA method to each pair, which in total results in $O(\sqrt{N}) \cdot \lceil d_{in}/\lfloor\sqrt{N}\rfloor\rceil = O(d_{in})$ homomorphic multiplications and $O(\sqrt{N}/d_{out}) \cdot \lceil d_{in}/\lfloor\sqrt{N}\rfloor\rceil = O(d_{in}/d_{out})$ homomorphic rotations.

When $d_{in} \geq d_{out} > \lfloor\sqrt{N}\rfloor$, Matrix Squaring splits A and B to derive $\lceil d_{in}/\lfloor\sqrt{N}\rfloor\rceil \cdot \lfloor d_{out}/\lfloor\sqrt{N}\rfloor\rfloor$ pairs of $\lfloor\sqrt{N}\rfloor \times \lfloor\sqrt{N}\rfloor$ matrices and $\lfloor\sqrt{N}\rfloor \times m$ matrices as well as $\lceil d_{in}/\lfloor\sqrt{N}\rfloor\rceil$ pairs of $(d_{out} \bmod \lfloor\sqrt{N}\rfloor)\times\lfloor\sqrt{N}\rfloor$ matrices and $\lfloor\sqrt{N}\rfloor\times m$ matrices, which totally needs $O(\sqrt{N})\cdot\lceil d_{in}/\lfloor\sqrt{N}\rfloor\rceil\cdot\lceil d_{out}/\lfloor\sqrt{N}\rfloor\rceil = O(d_{in}\cdot d_{out}/\sqrt{N})$ homomorphic multiplications and $O(\sqrt{N}/(d_{out} \bmod \lfloor\sqrt{N}\rfloor))\cdot\lceil d_{in}/\lfloor\sqrt{N}\rfloor\rceil = O(d_{in}/(d_{out} \bmod \sqrt{N}))$ homomorphic rotations. □

Proof of Lemma 6.1. To test a single model on a single sample, HESV and SecSV need to perform L matrix multiplications for the L linear layers. These matrix multiplications result in $O(\sum_{l=1}^{L}(d_{in}^{(l)} \cdot d_{out}^{(l)}/\sqrt{N}))$ HMults and $O(\sum_{l=1}^{L} \frac{d_{in}}{d_{out} \bmod \sqrt{N}})$ HRots for HESV while $O(\sum_{l=1}^{L} d_{in}^{(l)})$ $HMults$ and no HRot for SecSV.

Then, when testing a batch of test samples using the SIMD property of HE, we should decide on a batch size m that is compatible with all the layers since they may have different shapes. Therefore, we have $m \leq \min\{d_{in}^{(1)}, ..., d_{in}^{(L)}, \sqrt{N}\}$ for HESV and $m \leq \lfloor N/\max\{d_{out}^{(1)}, ..., d_{out}^{(L)}\}\rfloor$ for SecSV.

Therefore, to test a single model on M test samples, HESV needs $O(\frac{M\cdot(\sum_{l=1}^{L} d_{in}^{(l)}\cdot d_{out}^{(l)})}{\min\{d_{in}^{(1)},...,d_{in}^{(L)},\sqrt{N}\}\cdot\sqrt{N}})$ HMults and $O(\frac{M\cdot\sum_{l=1}^{L}\frac{d_{in}}{d_{out} \bmod \sqrt{N}}}{\min\{d_{in}^{(1)},...,d_{in}^{(L)},\sqrt{N}\}})$ HRots, while SecSV needs $O(\frac{M\cdot\max\{d_{out}^{(1)},...,d_{out}^{(L)}\}\sum_{l=1}^{L} d_{in}^{(l)}}{N})$ HMults and no HRot. Finally, since we have $O(2^n)$ models to test for each training round $t \in [T]$, we conclude that Lemma 6.1 stands true. □

Proof of Theorem 6.3. For any model $\theta \in [\theta_{S_1}, \theta_{S_2}]$, if $y = \arg\max_j f_\theta(x)[j]$, we have $f_\theta(x)[y] > f_\theta(x)[j], \forall j \neq y$. Because η is an entrywise strictly increasing function, we have

$$\forall\theta \in [\theta_{S_1}, \theta_{S_2}], (\theta^w x + \theta^b)[y] > (\theta^w x + \theta^b)[j], \forall j \neq y$$
$$\Rightarrow (\theta^w_{\{S_1,S_2\}} x + \theta^b_{\{S_1,S_2\}})[y] > (\theta^w_{\{S_1,S_2\}} x + \theta^b_{\{S_1,S_2\}})[j], \forall j \neq y$$
$$\Rightarrow f_{\theta_{\{S_1,S_2\}}}(x)[y] > f_{\theta_{\{S_1,S_2\}}}(x)[j], \forall j \neq y$$

from which we conclude that $\arg\max_j f_{\theta_{\{S_1,S_2\}}}(x)[j] = y$. □

Proof of Lemma 5.2. For each round t, for each client $i \in I^t$, , we have

$$\begin{aligned}
|\hat{\phi}_i^t - \phi_i^t| =& \Big|\sum_{S\subseteq I^t\backslash\{i\}} \frac{|S|!(|I^t| - |S| - 1)!}{|I^t|!}\big(\hat{v}(\theta^t_{S\cup\{i\}}) - \hat{v}(\theta^t_S)\big) \\
&- \sum_{S\subseteq I^t\backslash\{i\}} \frac{|S|!(|I^t| - |S| - 1)!}{|I^t|!}\big(v(\theta^t_{S\cup\{i\}}) - v(\theta^t_S)\big)\Big| \\
=& \Big|\sum_{S\subseteq I^t\backslash\{i\}} \frac{|S|!(|I^t| - |S| - 1)!}{|I^t|!}\big((\hat{v}(\theta^t_{S\cup\{i\}}) - v(\theta^t_{S\cup\{i\}})) \\
&- (\hat{v}(\theta^t_S) - v(\theta^t_S))\big)\Big| \\
=& \Big|\sum_{S\subseteq I^t\backslash\{i\}} \frac{|S|!(|I^t| - |S| - 1)!}{|I^t|}(\Delta v(\theta^t_{S\cup\{i\}}) - \Delta v(\theta^t_S))\Big| \\
\leq& \sum_{S\subseteq I^t\backslash\{i\}} \frac{|S|!(|I^t| - |S| - 1)!}{|I^t|}\big|(\Delta v(\theta^t_{S\cup\{i\}}) - \Delta v(\theta^t_S))\big| \\
\leq& \sum_{S\subseteq I^t\backslash\{i\}} \frac{|S|!(|I^t| - |S| - 1)!}{|I^t|}\big|\Delta v_{max}\big| = \Delta v_{max}
\end{aligned}$$

Therefore, for each client $i \in I$, we have

$$|\hat{\phi}_i - \phi_i| = |\sum_{t=1}^{T}(\hat{\phi}_i^t - \phi_i^t)| \leq \sum_{t=1}^{T}|\hat{\phi}_i^t - \phi_i^t| \leq \sum_{t=1}^{T}\Delta v_{max} = T\cdot\Delta v_{max}$$

□

Proof of Proposition 6.8. Under HESV (SecSV), the server(s) can only obtain encrypted models, encrypted (secret shares of) input features, encrypted output features, encrypted (secret shares of) labels, and the count of (the indices of) correctly predicted samples. Since the probabilities of the server(s) successfully inferring input features, output features, or model parameters from the count of (the indices of) correctly predicted samples are negligible, according to Definitions 6.5, 6.6, and 6.7, the server(s) cannot apply the defined attacks due to the lack of the necessary messages.

Let i_θ denote the owner of model θ. For each layer l, if $n \geq 3$, given that client i_l possseses the input features $X^{(l)}$, HESV and SecSV ensures that the output features $\hat{Y}^{(l)}$ are allocated to another client $i_{l+1} \neq i_l \neq i_\theta$. Since for all PPT adversaries $\mathcal{A}$ and for all linear layers l, there is a negligible ϵ such that $Pr[\mathcal{A}(\hat{Y}^{(l)}) = X^{(l)}] \leq \epsilon$ over all possible $(\theta^{(l)}, X^{(l)})$, we have

$$Pr[(\mathcal{A}(\hat{Y}^{(l)}) = X^{(l)}) \wedge (\mathcal{A}(X^{(l)}, \hat{Y}^{(l)}) = \theta^{(l)})]$$
$$=Pr[\mathcal{A}(X^{(l)}, \hat{Y}^{(l)}) = \theta^{(l)} | \mathcal{A}(\hat{Y}^{(l)}) = X^{(l)}]\cdot Pr[\mathcal{A}(\hat{Y}^{(l)}) = X^{(l)}] \leq \epsilon$$

Therefore, the probability of client i_{l+1} applying single-layer ES attacks to infer $\theta^{(l)}$ is negligible. Also, since for all PPT adversaries $\mathcal{A}$ and for all linear layers l, there is a negligible ϵ such that $Pr[\mathcal{A}(\hat{Y}^{(l)}) = \theta^{(l)}] \leq \epsilon$ over all possible $(\theta^{(l)}, X^{(l)})$, we have

$$Pr[\wedge_{l=1}^{l_2}((\mathcal{A}(\hat{Y}^{(l)}) = \theta^{(l)}) \wedge (\mathcal{A}(\hat{Y}^{(l)}) = X^{(l)})) \wedge (\mathcal{A}(\{\theta^{(l)}\}_{l=1}^{l_2}, \hat{Y}^{(l_2)}) = X^{(1)})] \leq \epsilon$$

Therefore, the probability of applying ES data stealing attacks to infer the features $X^{(1)} = X$ of test data is negligible.

Similarly, if $n \geq 4$, HESV and SecSV can ensure that client i_{L+1} who receives the output features $\hat{Y}^{(L)}$ is not the owner i_1 of the input $X^{(1)} = X$. Then, the probability of client i_{L+1} applying retraining attacks is negligible:

$$Pr[\wedge_{l=1}^{L}(\mathcal{A}(\hat{Y}^{(l)}) = X^{(l)}) \wedge (\mathcal{A}(X^{(1)}, \hat{Y}) = f')]$$
$$=Pr[\wedge_{l=1}^{L-1}(\mathcal{A}(\hat{Y}^{(l)}) = X^{(l)}) \wedge (\mathcal{A}(X^{(1)}, \hat{Y}) = f') | \mathcal{A}(\hat{Y}^{(L)}) = X^{(L)}]$$
$$\cdot Pr[\mathcal{A}(\hat{Y}^{(L)}) = X^{(L)}] \leq \epsilon$$

For each sample (x, y) in the batch, assume that it was in the training set $\mathcal{D}^{train}$ without loss of generality; the probability of client i_{L+1} applying an MI attack is negligible:

$$Pr[\wedge_{l=1}^{L}(\mathcal{A}(\hat{Y}^{(l)}) = X^{(l)}) \wedge (\mathcal{A}((x,y), \hat{Y}) = ((x,y) \text{ was in } \mathcal{D}^{train}))]$$
$$=Pr[\wedge_{l=1}^{L-1}(\mathcal{A}(\hat{Y}^{(l)}) = X^{(l)}) \wedge (\mathcal{A}((x,y), \hat{Y}) = ((x,y) \text{ was in } \mathcal{D}^{train})) | \mathcal{A}(\hat{Y}^{(L)}) = X^{(L)}]$$
$$\cdot Pr[\mathcal{A}(\hat{Y}^{(L)}) = X^{(L)}] \leq \epsilon$$

Finally, if $n \geq L+2$, HESV and SecSV can ensure that all layers of input and output features are possessed by different clients and that the model owner receives nothing. In this case, for all possible l_1, l_2 such that $l_1 < l_2$, the probability of client i_{l_2+1} applying multi-layer ES attacks to steal $\theta^{(l_1)}, ..., \theta^{(l_2)}$ is negligible:

$$Pr[\wedge_{l=l_1}^{l_2}(\mathcal{A}(\hat{Y}^{(l)}) = X^{(l)}) \wedge (\mathcal{A}(X^{(l_1)}, \hat{Y}^{(l_2)}) = (\theta^{(l_1)}, ..., \theta^{(l_2)}))]$$

$$=Pr[\wedge_{l=l_1}^{l_2-1}(\mathcal{A}(\hat{Y}^{(l)})=X^{(l)})\wedge(\mathcal{A}(X^{(l_1)},\hat{Y}^{(l_2)})=(\theta^{(l_1)},...,\theta^{(l_2)}))|\mathcal{A}(\hat{Y}^{(l_2)})=X^{(l_2)}]$$

$$\cdot\, Pr[\mathcal{A}(\hat{Y}^{(l_2)}) = X^{(l_2)}] \leq \epsilon$$

□

Table 7: RNN classifier for miRNA-mRNA.

GRU	Repeat the following steps 10 times:
	(1) Fully connects 64 input features to 32 input-reset features, fully connects 32 hidden features to 32 hidden-reset features, aggregates input-reset features and hidden-reset features, and calculates the sigmoid function of the aggregated result as reset gates: $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times64} \times \mathbb{R}^{64\times1}$, $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times32} \times \mathbb{R}^{32\times1}$, $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times1} + \mathbb{R}^{32\times1}$, $\mathbb{R}^{32\times1} \leftarrow Sigmoid(\mathbb{R}^{32\times1})$. (2) Fully connects 64 input features to 32 input-update features, fully connects 32 hidden features to 32 hidden-update features, aggregates input-update features and hidden-update features, and calculates the sigmoid function of the aggregated result as update gates: $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times64} \times \mathbb{R}^{64\times1}$, $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times32} \times \mathbb{R}^{32\times1}$, $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times1} + \mathbb{R}^{32\times1}$, $\mathbb{R}^{32\times1} \leftarrow sigmoid(\mathbb{R}^{32\times1})$. (3) Fully connects 64 input features to 32 input-new features, fully connects 32 hidden features to 32 hidden-new features, multiplies 32 reset gates with 32 hidden-new features to derive 32 reset features, and aggregates input-new features and reset features, and calculates the tanh function of the aggregated result as new gates: $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times64} \times \mathbb{R}^{64\times1}$, $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times32} \times \mathbb{R}^{32\times1}$, $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times1} \cdot \mathbb{R}^{32\times1}$, $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times1} + \mathbb{R}^{32\times1}$, $\mathbb{R}^{32\times1} \leftarrow tanh(\mathbb{R}^{32\times1})$. (4) Aggregate 32 hidden features and 32 new gates with weights of update gates and (1 - update gates) to update hidden features: $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times1} \cdot \mathbb{R}^{32\times1}$, $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times1} \cdot \mathbb{R}^{32\times1}$, $\mathbb{R}^{32\times1} \leftarrow \mathbb{R}^{32\times1} + \mathbb{R}^{32\times1}$.
FC	Fully connects 32 input features to 2 output features.

Table 8: DNN classifier.

Layer 1 (input)	Fully connects d input features to 100 output features: $\mathbb{R}^{100\times1} \leftarrow \mathbb{R}^{100\times d} \times \mathbb{R}^{d\times1}$
Layer $l \in \{2, L-1\}$ (hidden)	Fully connects 100 input features to 100 output features: $\mathbb{R}^{100\times1} \leftarrow \mathbb{R}^{100\times100} \times \mathbb{R}^{100\times1}$.
Layer L (output)	Fully connects 100 input features to c output features: $\mathbb{R}^{c\times1} \leftarrow \mathbb{R}^{c\times100} \times \mathbb{R}^{100\times1}$

Table 5: Logistic classifier for AGNEWS and BANK.

FC	Fully connects d input features to c output features: $\mathbb{R}^{2\times1} \leftarrow \mathbb{R}^{2\times48} \times \mathbb{R}^{48\times1}$
Sigmoid	Calculates the sigmoid function for each output feature

Table 6: CNN classifier for MNIST.

Conv.	Input image 28 × 28, kernel size 7 × 7, strize size 3, output channels 4: $\mathbb{R}^{4\times64} \leftarrow \mathbb{R}^{4\times28} \times \mathbb{R}^{28\times64}$
Square	Squares 256 output features
FC	Fully connects 256 input features to 64 output features: $\mathbb{R}^{64\times1} \leftarrow \mathbb{R}^{64\times256} \times \mathbb{R}^{256\times1}$
Square	Squares 64 output feautres
FC	Fully connects 64 input features to 10 output features: $\mathbb{R}^{10\times1} \leftarrow \mathbb{R}^{10\times64} \times \mathbb{R}^{64\times1}$

C DATASETS AND CLASSIFIERS

Datasets. We use the following datasets for experiments.

- **AGNEWS [85]**: It is a subdataset of AG's corpus news articles for news classification consisting of 120000 training samples and 7600 test samples with 4 classes of labels.
- **BANK [50]**: This dataset contains 41188 samples of bank customers' personal information for bank marketing. The goal is to predict whether a customer will subscribe a term deposit or not, which is a binary classification task. We split the dataset into 31188 samples for training and 10000 samples for testing.
- **MNIST [31]**: It is a dataset of handwritten digits and is one of the most famous benchmark for image classification. It has 60000 training images and 10000 test images, and the task is to identify the digit from 0 to 9 in a given image.
- **miRNA-mRNA [47]**: This dataset comprises human miRNA-mRNA pairs for miRNA target prediction, with 62642 pairs for training and 3312 pairs for testing. The task is to predict whether a given mRNA is the target of a given miRNA.

Classifiers. We use the following classifiers for experiments.

- **Logistic**: For the BANK and AGNEWS datasets, we use the logistic classifier with 1 fully-connected (FC) layer and the sigmoid activation (see Table 5).
- **CNN**: For the MNIST dataset, we follow [24] to use a convolutional NN (CNN) with 1 convolution layer and 2 FC layers with the square activation (see Table 6).
- **RNN**: For the miRNA-mRNA dataset, we adopt a recurrent NN named deepTarget [32] with 1 gate recurrent units (GRUs) layer and 1 FC layer (see Table 7).
- **DNN**: For all the datasets, we adopt the widely-used DNN classifier with 1 FC input layer, 1 FC output layer, and $L-2$ FC hidden layers with 100 hidden nodes (see Table 8).