Sphinx: Enabling Privacy-Preserving Online Learning over the Cloud

Han Tian[1] Chaoliang Zeng[1] Zhenghang Ren[1] Di Chai[1,2] Junxue Zhang[1,2] Kai Chen[1] Qiang Yang[1]
[1]Hong Kong University of Science and Technology [2]Clustar

2022 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-1316-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SP46214.2022.9833648

***Abstract*—With the growing complexity of deep learning applications, users have started to delegate their data and models to the cloud. Among these applications, online learning services, which involve both training and inference procedures, are widely deployed. To ensure privacy guarantee on the public cloud, researchers have proposed a plethora of privacy-preserving deep learning algorithms with different techniques, ranging from obfuscation mechanisms to cryptographic tools. However, none of them is applicable to online learning services. They either focus only on inference or training procedure while ignoring the other, or require non-colluding or trusted third parties.**

In this paper, we present Sphinx, an efficient and privacy-preserving online deep learning system without any trusted third parties. Sphinx strikes a balance between model performance, computational efficiency, and privacy preservation with systematical optimizations on both private inference and training protocols. At its core, Sphinx synthesizes homomorphic encryption and differential privacy reciprocally to maintain the model by keeping most of its parameters as plaintexts, enabling fast training and inference protocol designs. Meanwhile, by refining the homomorphic operation behaviors, Sphinx avoids most of the heavyweight homomorphic operations and minimizes the communication cost. As a result, Sphinx is able to reduce the training time significantly while achieving real-time inference without exposing user privacy. In our experiments, we find that compared to the pure homomorphic encryption solution, Sphinx is 35× faster for training and 4 orders of magnitude faster for inference, providing real-time inference response (0.05 seconds for MNIST and 0.08 seconds for CIFAR-10). Our experiments also demonstrate that Sphinx achieves promising model accuracy under a tight privacy budget (96% accuracy under $\epsilon = 2, \delta = 10^{-5}$ for MNIST) without a trusted data aggregator, and is more robust against practical reconstruction attacks.

I. Introduction

Although deep learning has become the fundamental infrastructure and core functionality for many applications, the computation requirement of decent training and fast inference on deep neural networks keeps increasing. To relieve the computation overhead, *machine-learning-as-a-service* (MLaaS) system is involved. The same trend happens in providing online learning services, which involves both training and inference procedures during the service time. For example, starting with a baseline model trained on a generic and large dataset such as ImageNet [1], the image classification service provider could provide inference services as well as fine-tuned personalized models for each user with their personal images.

Delegating the training and inference workloads to a public cloud inevitably raises privacy concerns. To solve this problem, previous wisdom has developed MLaaS systems with supports from various privacy-preserving techniques to protect user privacy (Section II). However, given the unique characteristic of online learning that requires both efficient inference and training procedures, none of the existing works is applicable to online learning services on deep neural networks. They either focus on providing inference service over encrypted data ([2]–[5]) that assumes a plain public model on the server, or concentrate on the training procedure on encrypted models ([6]–[10]) that leads to inefficient inference. Moreover, other potential solutions ([11]–[19]) make unrealistic assumptions about the threat model, such as multiple non-colluding servers or a trusted aggregator.

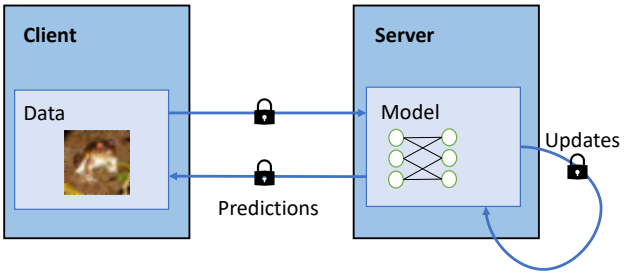


Fig. 1: A privacy-preserving online deep learning service with both training and inference services. Both the data and model parameters are protected against the service provider.

Therefore, we ponder a fundamental question: *as demonstrated in Figure 1, can we design an efficient and privacy-preserving online learning framework without additional third party requirements, e.g., non-colluding servers or a trusted aggregator?* Specifically, we focus on deep neural network, a widely used model in online learning services.

In this paper, we present Sphinx, an efficient and privacy-preserving online deep learning system offering both efficient training and real-time inference services on deep neural networks without a trusted party. Sphinx protects both model and input data, thus allowing users to delegate training and inference tasks to cloud servers without exposing data privacy. Sphinx is applicable for feed-forward neural networks consisting of fully connected layers and convolutional layers.

To achieve the above features, Sphinx uses differential privacy to bridge privacy-preserving training and private inference solutions. It also introduces new hybrid privacy-preserving protocols for training and inference phases, respectively, according to their specific characteristics and requirements. As far as we know, Sphinx is the first of its kind in the regime of privacy preserving deep learning to adopt different privacy-preserving protocols for both phases. To design privacy-preserving deep learning protocols by combining different primitives together, we have two-fold key technical challenges: i) how to design efficient and compatible training and inference protocols with different primitives respectively to fulfill their specific requirements, and ii) how to fully exploit

DOI 10.1109/SP46214.2022.00066

the possible merits from the hybrid protocols, in terms of both privacy and accuracy, by deliberately designing the privacy-preserving operation behaviors.

At its core, Sphinx divides each linear layer in deep neural networks into two parts: the linear component $\mathbf{W}$ and the bias component $\mathbf{b}$. Sphinx encrypts all the bias components with homomorphic encryption (HE), and perturbs the linear components with differential privacy (DP). We show that this design enables our high throughput training and low latency inference protocols (Section V), and builds a reciprocal relation between DP and HE techniques from both theoretical (Section VI) and empirical (Section IX) perspectives.

To accelerate the training procedure under the HE scheme, Sphinx makes several system optimizations (Section VII). First, by deliberately designing the homomorphic arithmetic operation behaviors between features, gradients and model parameters, it avoids most of the expensive rescaling and relinearization operations for ciphertext multiplications. Second, Sphinx accelerates the encryption operation and reduces the ciphertext size, which further lowers the communication time between the client and server.

Sphinx combines the above insights and optimizations to offer a reasonable privacy-accuracy trade-off, efficient training, and real-time inference. In the experimental section (Section IX), we show that Sphinx achieves $35\times$ less training time and $5\times$ lower communication costs for both neural networks on MNIST and CIFAR-10 datasets than the pure HE method. For inference, Sphinx achieves real-time inference in the online phase (0.05s for MNIST and 0.08s for CIFAR-10), which is *4 orders of magnitude* faster compared to the pure HE method. Moreover, given the same privacy budget, Sphinx achieves a similar model accuracy as the pure DP training algorithm DPSGD in [15] (96% accuracy under $\epsilon = 2, \delta = 10^{-5}$ for MNIST) without a trustworthy server.

One line of criticism of DP-based algorithms is the huge gap between the upper bound of privacy loss and the realistic privacy leakage in adversarial scenarios [20]. To verify the privacy guarantee provided by Sphinx, we evaluate the privacy-accuracy trade-off against current reconstruction attacks from gradients [21] (Section IX-E). The result shows that even with similar privacy cost compared to DP solutions, Sphinx can achieve a significantly stronger defense against attacks in a practical scenario.

II. Related Work

This section reviews existing privacy-preserving deep learning solutions that focused either on privacy-preserving inference or on privacy-preserving training but not both, which motivates our design of Sphinx. We provide a qualitative comparison between Sphinx and these works regarding the threat model and relative training and inference speeds approximated from their evaluation results in Table I.

Privacy-Preserving Inference on Neural Networks A number of solutions focus on the inference procedure, where the server holds a well-trained model to provide predictions-as-a-service for clients ([2]–[5], [24]–[28]). In this setting, since the model is either public or proprietary to the service provider, it is stored and evaluated as plaintext. Thus, the forward propagation only involves matrix multiplications between the encrypted input data and unencrypted model parameters, on which lots of optimizations have been proposed. For instance, Gazelle [3] introduced a hybrid secure inference algorithm, where the linear layers are evaluated with packed additively homomorphic encryption (PAHE) and the non-linear layers with garbled circuits. Based on Gazelle, Delphi [5] improves the inference latency significantly by bringing forward the heavyweight cryptographic computations to the preprocessing phase and developing a planner based on neural architecture search (NAS) to search for the most efficient approximation model meeting the target accuracy goal.

However, these optimizations for private inference cannot be transferred into online learning setting, as the training algorithm needs to encrypt the model parameters and gradients to protect training data against the cloud. Our work, on the other hand, is able to offer training and inference services simultaneously in a privacy-preserving way.

Privacy-Preserving Training on Neural Networks Solutions focusing on privacy-preserving training on neural networks can be classified into centralized and distributed, depending on the number of servers required. Some of the centralized training methods ([10], [22], [29]) adopt HE to encrypt both model weights and user data to outsource the training to the service provider. However, their training processes involve arithmetic operations between the encrypted neural network layers and encrypted features, which is an order of magnitude slower than the arithmetic operations between plaintexts and ciphertexts (Table II). Furthermore, these solutions do not take inference into consideration and simply apply the time-consuming forward phase in training for evaluation. Our work, however, provides a specific inference protocol based on lightweight secret sharing techniques, which is much faster, thus achieving real-time inference response.

Some other centralized training solutions adopt DP-based algorithms to protect privacy information by introducing randomization in the algorithm ([15]–[17], [19], [30]–[32]). For instance, Abadi et al. [15] proposed differentially private stochastic gradient descent (DPSGD) with Gaussian noise-additive mechanism and proved a tighter bound on the privacy guarantee. DP-FTRL improves DPSGD with tree aggregation trick to get rid of sampling and shuffling [19]. Instead of perturbing the gradients, some works proposed to introduce randomization into the objective function [16], the intermediate activation features [17], and the labels [31]. However, the trade-off between model performance and privacy level in DP-based algorithms has long been criticized in practice. Zhu at al. showed that to successfully defend privacy leakage from gradients, the model performance may drop significantly due to the noise added on gradients [33]. As a result, to preserve model performance, the privacy constraint companies and researchers used can hardly protect individual data ([34], [35]). Also, while focusing on publishing perturbed models,

	MiniONN [2]	Gazelle [3]	Delphi [5]	CryptoDL [22]	[10]	DPSGD [15]	DP-FTRL [19]	SecureML [11]	SecureNN [13]	FLASH [23]	SPHINX
Num of servers	1	1	1	1	1	1	1	2	3	4	1
Adversarial Model	1 P	1 P	1 P	1 P	1 P	No	No	1 P	MT	MT	1 P
Collusion	/	/	/	/	/	/	/	No	No	No	/
Inference	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Latency (relative)	10×	1×	1/10×	100×	100×	1/100×	1/100×	10×	1×	1/10×	1/10×
Train	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Throughput (relative)	/	/	/	1/100×	1/100×	100×	100×	1/10×	1×	10×	1×
Techniques	HE,GC,SS	HE,GC,SS	HE,GC,SS	HE	HE	DP	DP	HE,GC,SS	SS	SS	HE,DP,SS

TABLE I: A qualitative comparison between existing privacy-preserving deep learning outsourcing schemes and Sphinx. **A** and **P** denote active and passive adversarial capabilities. **MT** denotes honest majority setting, where most of the servers are assumed to be trustworthy. **HE**, **GC**, **SS** and **DP** stand for homomorphic encryption, garbled-circuits, secret sharing and differential privacy, respectively. The relative latency and throughput are approximated from their evaluation results.

DP-based algorithms do not protect the training and inference data against the service provider in the MLaaS setting. Furthermore, they assume a trusted data aggregator to apply noise and perform data aggregation, which is not needed in our work.

A number of works for distributed training solutions are based on MPC algorithms, where the computations on private data are distributed across multiple servers ([11]–[14], [23], [36]–[38]). For instance, SecureML [11] firstly proposed MPC alternatives to linear and non-linear functions to perform secure neural network training and inference. ABY3 [12] introduced the three-party computation version of ABY in [39] to improve efficiency, allowing switching between arithmetic, boolean, and Yao's GCs. SecureNN [13] applied three-party and four-party secure computation protocols to accelerate neural network secure training. FLASH [23] proposed a four-party privacy-preserving deep learning framework to achieve the strongest security notion of guaranteed output delivery, which allows one of the servers to be an active adversary. However, all of them assume multiple non-colluding servers or honest majority setting where the majority of the servers are trustworthy, and thus are impractical under the MLaaS setting.

There are also a number of works focusing on privacy-preserving federated learning, where multiple data owners collaboratively learn their local models without exposing user privacy ([40]–[52]). In the federated learning setting, the clients maintain their data and update the model locally. For instance, Shokri at al. proposed distributed selective stochastic gradient algorithm (selective SGD) and applied DP mechanisms on the uploaded gradients [40]. Meanwhile, Sinem Sav et al. employed HE to perform federated neural network learning across data owners with several cryptographic optimizations [48]. In a federated learning setting, clients are the data owners as well as main executors for the heavy computations of model training rather than service providers. As we are focusing on the MLaaS setting where the clients outsource the heavy computations to the service provider with large amounts of computation power, these works are beyond the scope of our paper.

III. Preliminaries

We provide basic knowledge about the threat model, deep neural networks, and privacy-preserving techniques, including HE and DP, that underlie Sphinx's training and inference protocols.

A. Threat Model

We consider a client-server model with two parties: client A and server B, where client A owns the data and server B provides training and inference services. In the training phase, A continuously provides collected input data, and B provides online training service to update the model. After following the privacy-preserving training protocol, B generates a well-trained model, which can be sent back to A or stored on the server to provide further secure inference service for the client. In the inference phase, the client provides the input data, and the service provider outputs the prediction result based on the trained model and the input data. During the training and inference procedures, B should not obtain any sensitive individual information from the input data. We consider a semi-honest server as in [2], [3], [5], [11]: it adheres to the protocol but is curious to infer privacy information from client A based on all the available information provided during the interaction.

B. DNN Training and Inference

Deep neural networks (DNNs), as a family of machine learning models, adopt stacked artificial neural network layers to extract features from raw data at different abstract levels hierarchically. With stacked layers, deep learning models can approximate arbitrary complex functions after training on collected massive data, and is capable of performing classification and regression tasks with higher accuracy and robustness compared with classical machine learning models [53]. Here, taking *convolutional neural networks* (CNNs) as an example, we focus on the abstraction of DNNs from a transformation perspective. A typical CNN processes a batch of input images through a sequence of neural network layers performing various transformations, including linear layer and non-linear layers. The CNN output can be probabilities for classification or prediction results for regression, based on the target task setting.

Linear layers. In a CNN, linear layers are of two types: convolutional (Conv) layers and fully-connected (FC) layers. Generally, input raw images are processed by several Conv layers to extract image features at different abstractions, which

are then flattened and fed to the FC layers to generate classification or regression results. Despite the differences in terms of structure, implementation and functionality, they both offer affine transformations involving linear transformations and translations.

Non-linear layers. The non-linear layers perform non-linear transformations on the input features. The most common ones in CNNs are max-pooling functions and activation functions. While activation functions such as ReLU and Sigmoid generally perform non-linear functions on the input features element-wise, the max-pooling function takes small chunks of input images or feature maps to perform sub-sampling.

C. Homomorphic Encryption

In our work, we focus on the CKKS HE scheme proposed in [54] and implemented in SEAL [55]. CKKS is a leveled homomorphic encryption supporting a limited number of additions and multiplications on ciphertexts based on the ring learning with error (RLWE) problem. Here we only give a brief introduction to CKKS and some operations (e.g., lazy rescaling, lazy relinearization) involved in our optimization section. For more details, please refer to [54].

1) Mathematical Background: To encrypt a message $m \in \mathbb{C}^{N/2}$, a vector of floating-point values, CKKS firstly encodes it into a plaintext r. The plaintext space is a polynomial ring $r \in \mathbb{Z}_q[X]/(\Phi_{2N}(X))$, where $\Phi_M(X)$ is the M-th cyclotomic polynomial and q the modulo of polynomial coefficients. Then with the public key p, CKKS encrypts a plaintext r with the public key into a ciphertext consisting of a pair of polynomials (c_0, c_1). To protect the preferred precision of messages during the encoding, CKKS multiplies the message by a scaling factor $\Delta > 0$ to keep precision of $1/\Delta$.

CKKS supports addition and multiplication between ciphertexts. More concretely, given two ciphertexts ct_1 and ct_2 from messages m_1 and m_2, we have $Dec(ct_1 + ct_2) \approx m_1 + m_2$ and $Dec(ct_1 * ct_2) \approx m_1 * m_2$. Also, CKKS supports operations between ciphertexts and plaintexts, which is much cheaper than the corresponding operations between ciphertexts.

Rescaling After multiplication (with ciphertext or plaintext), the scaling factor of the output ciphertext turns to Δ^2 and requires rescaling. CKKS performs rescaling by truncating a ciphertext into a smaller modulus q/Δ. For every ciphertext, the number of rescaling operation allowed is called the 'level' of the ciphertext (denoted by L in the paper), representing the number of multiplication/rescaling operations still allowed.

Relinearization After multiplication between two ciphertexts, the result ciphertext size will grow exponentially. To prevent it, CKKS performs relinearization to transform the ciphertext back to a pair of polynomials. [56] presents this relinearization technique in detail.

D. Differential Privacy

DP provides a theoretic privacy guarantee for randomized algorithms. It is a formal definition of privacy loss in the context of statistical and machine learning analysis. Through DP, data curators can release statistical results while still protecting individual information in the dataset.

Definition 1. *A randomized mechanism $M : D \to R$ with domain D and range R satisfies (ϵ, δ)-differential privacy if and only if for any two adjacent inputs $d, d' \in D$ and for any subset of outputs $S \subseteq R$, it holds that*

$$Pr[M(d) \in S] \leq e^{\epsilon} Pr[M(d') \in S] + \delta. \quad (1)$$

Adjacent inputs mean any two datasets that differ on only one row. With this definition, DP mathematically guarantees that the result of a DP analysis almost makes no difference to the analysis, whether or not any individual's private information is included in the input. To fulfill its definition, DP mechanism injects random noise into the original data or the statistical results, so deleting a single record from the original dataset will only generate a negligible effect on the algorithm output. Thus, we can obtain meaningful statistical results with acceptable accuracy loss. These mechanisms usually involve a trade-off between model performance and privacy level, depending on the noise.

IV. Combining Differential Privacy and Homomorphic Encryption

In this section, by diving into the characteristics of DP and HE, we make several key observations that motivate the combination of DP and HE in our design of Sphinx.

Differential privacy improves efficiency. As shown in Table II, we observe that the homomorphic arithmetic operations (multiplication and addition) take much less time when one operator is plaintext. As training and inference over DNNs involve a lot of matrix multiplications, by introducing DP to protect model parameters as plaintexts, we can significantly reduce the computation overhead in the training procedure compared to the pure HE methods. Furthermore, we observe that DP enables the lightweight techniques used in private inference works ([2], [3], [5]), which achieve low inference latency yet assume a public inference model in plaintext. In Sphinx, most of the model parameters are protected with DP mechanism and stored in plaintexts on the service provider. Thus, we can adopt the state-of-the-art cryptographic methods for private inference in Sphinx and significantly reduce the inference time in the online phase.

Secure computation improves privacy. We observe that we can achieve DP preservation without a trusted aggregator through secure computation schemes. Secure computation protocols ensure that the computation service provider executes the computation task without learning anything about the input data. For example, through HE, users can encrypt their data before sending them to the aggregator [57]. HE allows the aggregator to perform secure aggregation to get meaningful statistical results, upon which the differentially private noise-additive mechanism is applied. During the procedure, the individual privacy is protected by HE all the time against the aggregator. In this way, the combination of the two privacy-preserving methods enables centralized differentially private

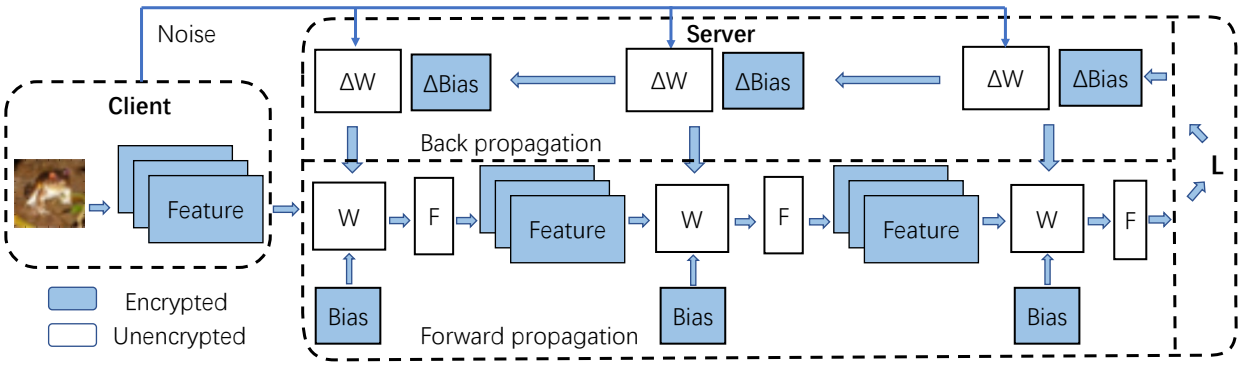


Fig. 2: The architecture of Sphinx. For an DNN model, the linear components W are perturbed with DP, and the bias components $Bias$ are encrypted with HE. F denotes the non-linear layers.

Operation	Runtime (μs)		
	$N = 2^{12}$ $L = 1$	$N = 2^{13}$ $L = 3$	$N = 2^{14}$ $L = 7$
Add	21	83	330
Add plain	10	42	166
Multiply	228	906	3682
Multiply plain	78	306	1204
Rescale	441	1894	8254
Relinearize	1257	6824	44273
Encode	414	1144	3926
Encrypt	2034	29947	20947
Decode	520	1922	8898
Decrypt	72	288	1293
Rotate one step	1297	6958	44616
Rotate random	5175	29057	196113

TABLE II: SEAL CKKS performance test for basic operations under λ=128-bit security. N: the cyclotomic polynomial degree. L: the 'level' of the ciphertexts.

training algorithms without the requirement of a trusted aggregator. Experiments in Section IX-D have shown that Sphinx, without a trusted server, achieves a comparable accuracy-privacy trade-off to the pure DP methods.

Masking defends attacks. We find that by hiding the bias parameters/gradients, we can effectively lower the risk of an attacker reconstructing data under DP mechanisms. Geiping et al. proved that to reconstruct the input of a biased fully-connected layer uniquely from the network's gradients, the knowledge of the derivatives w.r.t both the bias and the activation layer are essential [58]. By masking these gradients and activations against potential attackers, Sphinx effectively defends the reconstruction attacks under the same privacy level, as shown in Section IX-E.

Based on the above observations, we propose Sphinx, a privacy-preserving online learning framework for DNNs by combining HE with DP. In Sphinx, all the linear layers in DNNs are divided into two parts: the linear components and the bias components. Sphinx encrypts the bias components with HE, and perturbs the linear components with DP. We note that this design builds a reciprocal relation between DP and HE techniques:

- By leaving the linear components unencrypted and handled by DP mechanisms, we can avoid most of the expensive homomorphic ciphertext-ciphertext matrix operations in the training procedure;
- With the noisy linear components as plaintexts, we can adopt state-of-the-art secure inference protocols based on secret sharing in the inference phase to achieve real-time prediction response;
- With data and gradients encrypted, we can perform DP mechanisms to add noise on the aggregated gradient with no need to trust the cloud;
- Masked neural network layers make it practically harder for attackers to spy on the input features.

V. Design

This section describes the design of Sphinx in detail. We illustrate the model architecture in Figure 2. Given one input sample, the forward propagation operation for each convolutional or fully connected layer in feed-forward neural networks can be formalized as:

$$\mathbf{a_{i+1}} = f(\mathbf{W_i}^\intercal \mathbf{a_i} + \mathbf{b_i}), \tag{2}$$

where $\mathbf{a_i}$ represents the input vector of the i-th layer ($\mathbf{a_1}$ is the input data), $\mathbf{W_i}^\intercal \mathbf{a_i} + \mathbf{b_i}$ is the affine transformation of the input feature, and f denotes the composite function including operations such as pooling and non-linear activation functions. We separate the affine transformation into the linear part which involves $\mathbf{W_i}$, and the translation part that contains $\mathbf{b_i}$. We call $\mathbf{W} = (\mathbf{W_1}, \mathbf{W_2}, ...\mathbf{W_K})$ the **linear components** and $\mathbf{b} = (\mathbf{b_1}, \mathbf{b_2}, ...\mathbf{b_K})$ the **bias components** in the rest of the paper, where K denotes the number of linear layers in the deep neural networks. In this section, we will explain how to adopt HE and DP in the two components, respectively, to harvest the benefits from both techniques.

The design can be adopted for convolutional neural networks (CNN) and multilayer perceptron (MLP). Here we focus on the convolutional neural networks containing convolutional layers, non-linear activations, pooling layers, and fully connected layers.

A. Training Phase

We observe from Table II that the homomorphic matrix multiplication between a ciphertext and a plaintext is much faster

Algorithm 1: Sphinx Training Algorithm

Input : *Client*: Training dataset X with size N, noise scale σ, gradient norm bound C; *Server*: Model M with K layers, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_j \mathcal{L}(\theta, \mathbf{x_j})$, learning rate η_t, batch size B

Output: Trained model parameters $\theta^T = \{\mathbf{W}^T, [\mathbf{b}^T]\}$

The client creates an encryption key pair (PK, SK) and send the public key to the server;

The server initializes the plaintext linear components $\mathbf{W}^0$ and the encrypted bias components $[\mathbf{b}^0]$ with PK;

for $t \leftarrow 0$ **to** $T-1$ **do**

Client:

Takes a random sample batch $\mathcal{B} = \{\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x_B}\}$ with sampling probability B/N, encrypt them with PK and send the encrypted batch to the server;

Server:

Computes the gradients for the linear components $[\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{x_j})]$ and bias components $[\nabla_{\mathbf{b}}\mathcal{L}(\mathbf{x_j})]$ according to Equation 3,4 for each sample $\mathbf{x_j}$;

Computes the average gradients of bias components: $[\overline{\mathbf{g}}_{\mathbf{b}}] \leftarrow \frac{1}{B}(\sum_j [\nabla_{\mathbf{b}}\mathcal{L}(\mathbf{x_j})])$;

Updates the bias components: $[\mathbf{b}^{t+1}] \leftarrow [\mathbf{b}^t] - \eta_t[\overline{\mathbf{g}}_{\mathbf{b}}]$;

Sends $[\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{x_j})]$ back to the client;

Client:

Decrypts $[\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{x_j})]$ with SK;

Clips the gradient: $\nabla_{\mathbf{W}}\widetilde{\mathcal{L}}(\mathbf{x_j}) \leftarrow \max(1, \frac{\|\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{x_j})\|_2}{C})$;

Computes the average gradients of linear components: $\overline{\mathbf{g}}_{\mathbf{W}} \leftarrow \frac{1}{B}(\sum_j \nabla_{\mathbf{W}}\widetilde{\mathcal{L}}(\mathbf{x_j}))$;

Adds noise: $\widetilde{\mathbf{g}}_{\mathbf{W}} \leftarrow \overline{\mathbf{g}}_{\mathbf{W}} + N(0, \frac{\sigma^2 C^2}{B^2}\mathbf{I})$;

Sends $\widetilde{\mathbf{g}}_{\mathbf{W}}$ back to the server;

Server:

Update the linear components: $\mathbf{W}^{t+1} \leftarrow \mathbf{W}^t - \eta_t\widetilde{\mathbf{g}}_{\mathbf{W}}$;

than that between two ciphertexts. To exploit such performance characteristic, for each layer, we extract and encrypt the bias component $\mathbf{b}$ with HE, leaving the linear component $\mathbf{W}$ perturbed with DP later. Here we show how this modification avoids most of the computationally intensive operations and improves training efficiency while still protecting the input data and model functionality against the server.

During the training procedure, the client encrypts the training data and then sends it to the server as the encrypted input of the first layer in the model. Given a ciphertext $[u]$ and a plaintext v, the scalar multiplication result is also a ciphertext $[u*v]$. Thus, for every layer in the neural networks, the forward propagation operation can be formalized as

$$[\mathbf{a_{i+1}}] = f(\mathbf{W_i}^\intercal[\mathbf{a_i}] + [\mathbf{b_i}]), \tag{3}$$

where only scalar multiplication and homomorphic addition operation occur: the homomorphic matrix multiplication between ciphertexts is avoided with $\mathbf{W_i}$ unencrypted.

The forward propagation continues layer by layer until it produces the cost function $\mathcal{L}(\theta)$. To minimize it using gradient descent, we need to adopt the backpropagation algorithm to calculate the gradients of the cost function $\nabla_\theta\mathcal{L}(\theta)$ with respect to the model parameters θ, including $\mathbf{W}$ from the linear components and $\mathbf{b}$ from the bias components. For one layer described in Equation 2, starting from the gradient with respect to the output of the layer $\nabla_{\mathbf{a_{i+1}}}\mathcal{L}$, the backpropagation computation consists of two parts: (i) computing the gradients on the current layer ($\nabla_{\mathbf{b_i}}\mathcal{L}$ and $\nabla_{\mathbf{W_i}}\mathcal{L}$), and (ii) propagating the gradients back to the previous layer ($\nabla_{\mathbf{a_i}}\mathcal{L}$). As the propagated gradients are encrypted, the backpropagation operation in each layer can be formalized as follows:

$$\begin{aligned}
[\nabla_{\mathbf{b_i}}\mathcal{L}] &= [\nabla_{\mathbf{a_{i+1}}}\mathcal{L}] \odot f'(\mathbf{W_i}^\intercal[\mathbf{a_i}] + [\mathbf{b_i}]) \\
[\nabla_{\mathbf{W_i}}\mathcal{L}] &= [\mathbf{a_i}]([\nabla_{\mathbf{a_{i+1}}}\mathcal{L}] \odot f'(\mathbf{W_i}^\intercal[\mathbf{a_i}] + [\mathbf{b_i}]))^\intercal \\
[\nabla_{\mathbf{a_i}}\mathcal{L}] &= \mathbf{W_i}([\nabla_{\mathbf{a_{i+1}}}\mathcal{L}] \odot f'(\mathbf{W_i}^\intercal[\mathbf{a_i}] + [\mathbf{b_i}])).
\end{aligned} \tag{4}$$

We can see from the formulas that with the linear components $\mathbf{W}$ unencrypted, while the gradient calculation for each layer ($\nabla_{\mathbf{b_i}}\mathcal{L}$ and $\nabla_{\mathbf{W_i}}\mathcal{L}$) stays the same as that for the fully encrypted model, the homomorphic matrix multiplications between ciphertexts are avoided during the gradient propagation, i.e., calculating $\nabla_{\mathbf{a_i}}\mathcal{L}$.

The server trains neural networks with the well-known stochastic gradient descent (SGD) algorithm, iterating on mini-batches of the training dataset to update model parameters. Algorithm 1 outlines Sphinx's training algorithm. During the process, the linear component $\mathbf{W_i}$ for each layer is perturbed with DP via introducing additive noise on their gradients. At each step, the client takes a random sample batch with size B, encrypts it with the generated public key, and sends it to the server as the input. Based on the input batch, the server computes the encrypted gradients of all the model parameters following the forwarding and back-propagation formulas described above. For the bias components, the server aggregates and averages the gradients locally, and directly updates the model parameters $[\mathbf{b}]$. For the linear components, the server sends their gradients $[\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{x_j})]$ back to the client for decryption. Upon receiving the encrypted gradients, the client decrypts them with the private key and clips the gradients to ensure that $\left\|\nabla_{\mathbf{W}}\widetilde{\mathcal{L}}(\mathbf{x_j})\right\|_2 \leq C$, thus guaranteeing that the noise added later is sufficient to meet the target DP level. Finally, the client adds Gaussian noise on the averaged gradients and sends perturbed $\widetilde{\mathbf{g}}_{\mathbf{W}}$ back to the server to update the model.

B. Inference Phase

In the inference phase, the client sends the encrypted samples to the service provider for prediction or classification requests. We design a specific protocol for inference based on the following insights: i) while the training data are prepared beforehand in mini-batches, inference samples often come in a real-time stream requiring immediate feedback,

and ii) in Sphinx, the updated model contains plain noisy linear components. Thus, we can utilize the secret sharing based technique used in private inference protocols on linear operations [5], which assumes a public plaintext model on the server. Here we give an overview of our protocol.

On the server, Sphinx converts the linear components $\mathbf{W}$ into fixed-point representations and embeds them into a prime finite field $\mathcal{F}$. Once the model is updated, Sphinx takes a preprocessing phase to generate secret shares between the client and the server for each layer i:

1) The client and the service provider respectively generate random masking vectors $\mathbf{r_i}, \mathbf{s_i}$ in $\mathcal{R}^n$ and $\mathcal{R}^m$, where n, m are the sizes of the input and output in i-th linear layer.
2) The client encrypts $\mathbf{r_i}$ with HE scheme and sends $[\mathbf{r_i}]$ to the server. The server then computes $[\mathbf{W_i} \cdot \mathbf{r_i} + \mathbf{b_i} - \mathbf{s_i}]$ and sends it back to the client.
3) The client decrypts it and obtains $(\mathbf{W_i} \cdot \mathbf{r_i} + \mathbf{b_i} - \mathbf{s_i})$. Thus the client and the server hold an additive secret sharing of $\mathbf{W_i r_i} + \mathbf{b_i}$ for each layer.

In the online phase, once an input sample $\mathbf{x}$ comes, Sphinx executes the following steps starting with $i = 1$:

1) The client calculates $\mathbf{x} - \mathbf{r_i}$ and sends it to the server, and the server calculates $\mathbf{W_i}(\mathbf{x} - \mathbf{r_i}) + \mathbf{s_i}$. Since the client has $(\mathbf{W_i} \cdot \mathbf{r_i} + \mathbf{b_i} - \mathbf{s_i})$, the client and the server now hold an additive secret sharing of $\mathbf{W_i x} + \mathbf{b_i}$.
2) The server sends $\mathbf{W_i}(\mathbf{x} - \mathbf{r_i}) + \mathbf{s_i}$ to the client who then adds it with the local share to obtain $\mathbf{W_i x} + \mathbf{b_i}$, and performs non-linear operations on it.
3) The client repeats the same procedure for the next layer.

In the inference algorithm, the encrypted bias components are handled in the preprocessing phase, and all the arithmetic operations in the online phase are performed over prime fields.

We note that private inference protocols generally perform non-linear layers using MPC techniques such as garbled circuits to protect the intermediate features against both parties [3], [5]. In our online learning setting, the training data also comes from the client. Thus, Sphinx has no need to protect the non-linear function results against the client and avoids heavyweight cryptographic schemes.

To handle n inference samples, Sphinx needs n different additive secret shares, which can also be prepared in mini-batches to accelerate the preprocessing phase, as homomorphic ciphertexts in CKKS are efficient for SIMD computations [54].

VI. Privacy Analysis

We now provide privacy analysis for Sphinx. Essentially, Sphinx has different levels of privacy preservations of user data for the training and inference phases.

For the training phase of Sphinx, we adopt DP to provide theoretical privacy guarantees. Our training protocol can be viewed as a *homomorphic* version of DPSGD [15] without a trusted server. It adopts the Sampled Gaussian mechanism (SGM) that samples mini-batches from the dataset and adds Gaussian noise on the aggregated gradients w.r.t. the linear components. Sphinx enables centralized differentially private training without a trusted server for several reasons: i) The input data sent is protected by HE, thus not exposed to the server; ii) the gradients calculated on the server side are in encrypted form so that individual gradients are not exposed to the cloud server, and iii) the noise-additive mechanism is applied to the aggregated gradients on the client side after decryption. Thus, the random number generator is preserved against the server. During the training, only the perturbed aggregated gradients of the linear components are exposed to the server, which conforms to the threat model used in DP-based methods ([15]–[17], [19], [31], [32]) that the DP mechanism protects both the final model and the aggregated gradients in mini-batches.

The DP mechanism in Sphinx is an instance of *adaptive composition*, where we sequentially introduce noise on the gradients for each training step. In DP analysis, the privacy loss is defined as the random variable measuring the difference between the two probability distributions before and after adding noise. Specifically, for two adjacent inputs d and d', the privacy loss of a DP mechanism $\mathcal{M}$ at the outcome o is defined as:

$$c\left(o; \mathcal{M}, d, d'\right) \triangleq \log \frac{\Pr[\mathcal{M}(d) = o]}{\Pr\left[\mathcal{M}\left(d'\right) = o\right]}. \tag{5}$$

With more and more aggregated gradients exposed to the server during the training process, the privacy loss accumulates. In our training protocol, we adopt the **moments accountant** method proposed in [15] to estimate and trace the bound of the accumulated privacy loss. The moments accountant performs composition on the log moments bounds, which can lead to a tighter bound compared to the strong composition theorem in [59]. Given two adjacent inputs d, d' and the DP mechanism M, the λ^{th} moment is defined as:

$$\alpha_{\mathcal{M}}\left(\lambda; d, d'\right) \triangleq \log \mathbb{E}_{o \sim \mathcal{M}(d)}\left[\exp\left(\lambda c\left(o; \mathcal{M}, d, d'\right)\right)\right]. \tag{6}$$

[15] has proven the composability of moments bound and how it is related to the (ϵ, δ)-DP. Thus in Sphinx, the moments accountant estimates the bound of $\alpha_{\mathcal{M}}(\lambda)$ at each learning step and sums them all to estimate the overall bound, which can be converted to the accumulated privacy cost during the training procedure, i.e., the current (ϵ, δ)-DP guarantee. The bound on the privacy loss of Sample Gaussian mechanism has been studied in several works ([15], [60], [61]). Based on the asymptotic bound proof in DPSGD [15], we have the DP guarantee of our training protocol. Due to the post-processing immunity property of DP, calculations on the model in the inference phase will not incur any further privacy leaks on the training data.

Theorem 1. *Given the training batch size m, the number of training steps T, and the size of the training dataset N, these exists constants c_1, c_2 that, $\forall \epsilon < c_1 \frac{m^2}{N^2} T, \forall \delta > 0, \forall \sigma \geq c_2 \frac{m\sqrt{Tlog(1/\delta)}}{\epsilon N}$, Algorithm 1 is (ϵ, δ)-DP.*

For the inference phase, we adopt the private inference technique based on secret sharing. Here we prove that the

inference protocol gives the privacy guarantee that the service provider cannot infer any useful information about the evaluated data. Specifically, for the inference protocol Π between the server holding neural network model parameters $\mathbf{M} = (\mathbf{W_1}, \mathbf{W_2}, .., \mathbf{W_K}, [\mathbf{b_1}], [\mathbf{b_2}], ...[\mathbf{b_K}])$ and the client having the evaluated input data $\mathbf{x}$, we prove that there exists an efficient simulator Sim such that the view of of the server executing Π is computationally indistinguishable from the output of the simulator, i.e., $\mathsf{View}_S^{\Pi} \approx_c \mathsf{Sim}_S(\mathbf{M})$.

We give the simulator Sim that proceeds as follows:

1) Sim initializes the server with a uniform random tape. It chooses a public key PK for the HE scheme and sends PK to the server.
2) In the preprocessing phase, once the model is updated, Sim encrypts $\mathbf{0}$, sends $[\mathbf{0}]$ to the server, and receives the returned result for each layer.
3) In the online phase, once an input sample $\mathbf{x}$ comes, Sim chooses a uniformly chosen $\mathbf{r}$, sends it to the server, and receives the returned result for each layer.

In the preprocessing phase, instead of sending the random mask $[\mathbf{r_i}]$, the simulator sends $[\mathbf{0}]$ to the server for each layer. It follows from the semantic security of HE that they are computationally indistinguishable. In the online phase, instead of sending masked input $\mathbf{x} - \mathbf{r_i}$, the simulator sends uniformly chosen value from $\mathcal{R}$. They are both uniformly distributed and thus computationally indistinguishable.

As a result, the real world distribution is computationally indistinguishable from the simulated distribution via a simple hybrid argument. As the simulator does not use any information about the evaluated input $\mathbf{x}$, the server learns nothing in the real world.

VII. Optimizations

We present several system optimizations for Sphinx, which fully exploit the possible merits of the combination of cryptographic tools and DP techniques to further accelerate the encrypted model training procedure.

A. Fast Homomorphic Multiplication

To enable further multiplication, multiplication operation between a ciphertext and a plaintext are followed by rescaling, and that between two ciphertexts are followed by both rescaling and relinearization. Inspired by previous HE-based methods seeking to avoid heavy rescaling and relinearization with *lazy rescaling* [26] and *lazy relinearization* [62] techniques, we deliberately design multiplication behaviors between model parameters, input features and gradients in neural network layers. We show that with the linear components as plaintexts, Sphinx can eliminate most of the rescaling and relinearization operations during the training procedure[1], which further enables it to adopt smaller encryption parameters with a shallower multiplicative depth, leading to less computation time and communication cost. In particular, while the pure

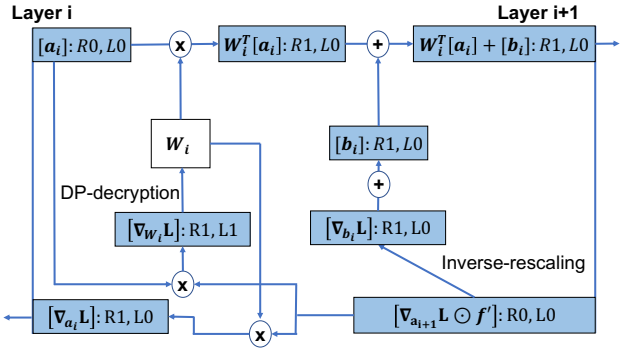


Fig. 3: The arithmetic operation behaviors between two layers in Sphinx.

HE method requires at least three levels ($L = 3, N = 2^{13}$) for rescaling operations during the training procedure[2], Sphinx allows the encryption parameters ($L = 1, N = 2^{12}$) with only one level.

For every ciphertext, we set two flags (Rx, Lx) to represent whether the ciphertext needs rescaling or relinearization for further multiplication. Fresh encrypted ciphertexts have the flags $(R0, L0)$, which means that they are ready for multiplication. The resulting ciphertext of multiplication between two ciphertexts has the flags $(R1, L1)$, requiring rescaling and relinearization operations to enable further multiplication. The resulting ciphertext of multiplication between a ciphertext and a plaintext has the flags $(R1, L0)$, requiring only rescaling.

Based on the design in Section V, we illustrate the detailed arithmetic operation behaviors for each layer in Sphinx in Figure 3. In the forward propagation, the activations or input features $[\mathbf{a_i}]$ fed into layer i are multiplied and aggregated with the plaintext linear component $\mathbf{W_i}$ with no rescaling needed. The reason is that homomorphic addition can be performed among the ciphertexts with the same augmented scaling factor and ciphertext size. In particular, we can perform addition on two ciphertexts if they both have the flags $(R0, L1)$, $(R1, L0)$, or $(R1, L1)$. We maintain our encrypted bias components with the flags $(R1, L0)$, thus the addition between $\mathbf{W}_l^T[\mathbf{a_i}]$ and $[\mathbf{b_i}]$ can be proceed without rescaling. The result is then fed into the non-linear activation function.

For backpropagation, given the gradients from the next layer $i + 1$, the procedure follows Equation 4. The gradients of the bias components $[\nabla_{\mathbf{b_i}}\mathcal{L}]$ can be obtained directly from $[\nabla_{\mathbf{a_{i+1}}}\mathcal{L}] \odot f'(\mathbf{W_i}^\intercal \mathbf{a_i} + \mathbf{b_i})$ and have the flags $(R0, L0)$. In order to update $\mathbf{b_i}$ efficiently, instead of performing rescaling on $[\mathbf{b_i}]$, we perform an *inverse-rescaling* operation on the gradients $[\nabla_{\mathbf{b_i}}\mathcal{L}]$ to increase the scaling factor of ciphertexts by simply performing scalar multiplication with $\mathbf{1}$, which is much faster. The *inverse-rescaled* gradients have the flags $(R1, L0)$ now and can be updated into $[\mathbf{b_i}]$ without further rescaling.

For the computation of gradients with respect to the linear components $[\nabla_{\mathbf{W_i}}\mathcal{L}]$ and the propagated gradients $[\nabla_{\mathbf{a_i}}\mathcal{L}]$,

[1]Except for those used in the rotation-based gradient aggregation over the encrypted bias components in our algorithm, which consume at least one 'level' of ciphertext.

[2]The encrypted linear layers require at least 3 levels: two levels for the rescaling operations after matrix multiplication in both FP and BP phases, and one for rotation-based aggregation.

the result goes back to the client for decryption and DP mechanism (*DP-decryption* in Figure 3) to avoid further rescaling and relinearization operations. Directly performing decryption without rescaling and relinearization also results in more accurate plaintexts, because rescaling performs rounding on ciphertexts and relinearization introduces noise. We also note that without relinearization, the size of the gradients of linear components $[\nabla_{\mathbf{W_i}}\mathcal{L}]$ will increase, resulting in a higher communication cost. However, because the bottleneck of *DP-decryption* is the ciphertext decryption on the client side and Sphinx conducts the communication in an asynchronous fashion, the overall training runtime will not increase.

B. Accelerating Client-Server Communication

Sphinx adopts an interactive protocol between the client and the cloud server mainly for two purposes: i) to perform the DP aggregation on the gradients of model linear components as shown in Algorithm 1, and ii) to evaluate the non-linear activation functions in plaintext on the client side for both training and inference phases, which are not supported by the current HE schemes. Also, the decryption and re-encryption on the client-side work as client-aided bootstrapping to refresh ciphertexts to allow more arithmetic operations [3]. The interactive protocol is straightforward: the server sends the ciphertexts back to the client for evaluation. The client decrypts and evaluates them in plaintext and sends them back to the server after either re-encryption or DP mechanism as shown in Algorithm 1.

To reduce the communication cost between the two parties, we implement the following optimizations:

Forward propagation cache. In the training procedure, to backpropagate the gradients across layers, i.e. calculating $[\nabla_{\mathbf{a_{i+1}}}\mathcal{L}] \odot f'(\mathbf{W_i}^\intercal\mathbf{a_i} + \mathbf{b_i})$, the gradients of the non-linear activation functions $f'(\mathbf{W_i}^\intercal\mathbf{a_i} + \mathbf{b_i})$ are involved. Instead of sending these encrypted activation functions back to the client for decryption and gradient calculation in the backpropagation phase, we realize that it is more efficient to calculate them beforehand in the forward propagation to avoid duplicate computations and communications. Thus, we design the forward propagation cache in Sphinx: In the forward propagation phase, the client caches the gradients of each non-linear layer locally as plaintexts, which can later be used in the backpropagation phase. For max-pooling layers, the client caches the mapping between the input features and subsampled output features. Thus, both the maximum feature values and the subsampled indices are protected against the service provider.

Zero decryption and zero encryption. In the training protocol, decryption and re-encryption occur in every ciphertext sent to the client, for which we realize that no more arithmetic operations are needed. Thus, Sphinx adopts *zero decryption* to reduce the level of prepare-to-send ciphertexts to 0, resulting in a much smaller ciphertext size during the communication. Table III shows the ciphertext sizes under different encryption parameters.

	$N = 2^{12}$		$N = 2^{13}$		$N = 2^{14}$	
	$L = 1$	$L = 0$	$L = 3$	$L = 0$	$L = 7$	$L = 0$
Size (MB)	0.13	0.07	0.52	0.13	2.10	0.26

TABLE III: The ciphertext sizes under different cyclotomic polynomial degrees and levels.

For the re-encryption part on the client size, we realize that homomorphic addition between ciphertext and plaintext is much faster than the encryption operation, as shown in Table II. Thus, the client in Sphinx adopts an encryption method we called *zero encryption*. In the offline preprocessing phase, the client prepares a stream of *zero ciphertexts* by encrypting $\mathbf{0}$, which works as one-time pad ciphertexts. Once the client needs to encrypt a message, it encrypts the plaintext by directly adding it with a *zero ciphertext* and sends it to the server, thus shortening the encryption time in the online phase.

VIII. System Implementation

We implement Sphinx, our privacy-preserving online learning framework in C++. We use SEAL 3.6 as the homomorphic encryption library[3]. Inspired by TenSEAL[4], a library supporting homomorphic encryption operations on high-dimensional tensors, we implement the homomorphic versions of the common operations in neural networks layers, such as convolution, pooling, and dot-product. Following [26] and [24], we pack our data and activation features in the batch axis (batch-axis packing) during the training. For example, for an input image, Sphinx stores a 3D ciphertext tensor of shape (C,H,W), removing the batch dimension, which is much efficient for training over mini-batches. Sphinx's deep learning framework implementation is based on KANN[5], a lightweight library allowing efficient inference and training on neural networks including MLP, CNN, and RNN. We implement our inference protocol based on Delphi[6][5]. Furthermore, we adopt the **moment accountant** implemented in [60] to keep track of the privacy loss during the learning process[7].

IX. Evaluation

In this section, we evaluate the performance of Sphinx with respect to both computation efficiency and privacy preservation. Our results reveal that:

- For training, Sphinx achieves 35× less training time and 5× lower communication cost for both neural networks on MNIST and CIFAR-10 datasets than pure HE methods (Section IX-B).
- For inference, Sphinx achieves real-time inference in the online phase (0.05s for MNIST and 0.08s for CIFAR-10), which is $5.8{*}10^4\times$ and $4.2{*}10^5\times$ faster compared to pure HE methods, respectively. Moreover, the communication cost of Sphinx is $1.9 * 10^5\times$ and $3.6 * 10^4\times$ lower for both tasks (Section IX-C).

[3]https://github.com/microsoft/SEAL
[4]https://github.com/OpenMined/TenSEAL
[5]https://github.com/attractivechaos/kann
[6]https://github.com/mc2-project/delphi
[7]https://github.com/tensorflow/privacy

- Given the same privacy budget, Sphinx achieves comparable model accuracy compared with the pure DP training algorithm DPSGD [15], without a trusted server (Section IX-D).
- We observe that by masking the bias components, Sphinx effectively defends the reconstruction attack from exposed gradients, compared to the pure DP algorithm (Section IX-E).

A. Evaluation Setup

We evaluate Sphinx on two physical machines for the two parties, each with 40 Intel Xeon CPU E5-2683 v4 cores at 2.1GHz and 128GB memory. The two machines are connected in LAN networking using a 10Gbps link. All experiments are conducted under Ubuntu 18.04.5 LTS, and our library is compiled using GCC 7.5.0 with -O2 optimization setting.

In following experiments, we use two image datasets:

1) The MNIST dataset contains 70,000 28*28 grayscale images for handwritten digits from '0' to '9', 60,000 for training, and 10,000 for testing. The task is to classify the correct handwritten digit in the given image.
2) CIFAR-10 is another image dataset containing 60,000 32×32 color images (with 3 channels), 50,000 for training and 10,000 for testing. The images in CIFAR-10 are classified into 10 classes, including birds, cats, planes, etc. The task is to classify the correct label for the given color image. The images in CIFAR-10 are more complicated with more labels and channels, thus requiring deeper neural networks.

We implement and train the two neural network architectures (Figure 4 and Figure 5) for each dataset similar to [2]. We add the bias components after the linear operation for each convolutional layer. The best accuracy we achieve for both non-private models on MNIST and CIFAR-10 is 98.5% and 78.8%, respectively. To avoid the influence of thread synchronization, we use a single thread for the server and the client for runtime measurements in Section IX-B and IX-C.

1) *Input*: $\mathbb{R}^{28\times28}$
2) *Convolution*: window size 5×5 with stride (1,1), number of channels 16, no pad, with bias. Outputs: $\mathbb{R}^{16\times24\times24}$
3) *ReLU*: ReLU non-linear function. Outputs: $\mathbb{R}^{16\times24\times24}$
4) *Average Pooling*: window size 1×2×2, stride (2,2). Outputs: $\mathbb{R}^{16\times12\times12}$
5) *Convolution*: window size 5×5 with stride (1,1), number of channels 16, no pad, with bias. Outputs: $\mathbb{R}^{16\times8\times8}$
6) *ReLU*: ReLU non-linear function. Outputs: $\mathbb{R}^{16\times8\times8}$
7) *Average Pooling*: window size 1×2×2, stride (2,2). Outputs: $\mathbb{R}^{16\times4\times4}$
8) *Fully Connected*: hidden neuron number 100. Outputs: $\mathbb{R}^{100\times1}$
9) *ReLU*: ReLU non-linear function. Outputs: $\mathbb{R}^{100\times1}$
10) *Fully Connected*: hidden neuron number 10. Outputs: $\mathbb{R}^{10\times1}$

Fig. 4: The neural network architecture for the MNIST dataset.

B. Training with Sphinx

For training, we set the batch size $B = 500$ for the model in Figure 4 on MNIST, and $B = 2000$ for the model in Figure 5 on CIFAR-10. The training data is shuffled, batched, encrypted on the client and then sent to the cloud. We adopt the stochastic gradient descent (SGD) method and set the learning rates to be 0.1 and 0.05 for both tasks. For CIFAR-10 task, we initialize the convolutional layers with the pre-trained weights trained on CIFAR-100 dataset.

1) *Input*: $\mathbb{R}^{3\times32\times32}$
2) *Convolution*: window size 5×5 with stride (1,1), number of channels 64, pad (2,2), with bias. Outputs: $\mathbb{R}^{64\times32\times32}$
3) *ReLU*: ReLU non-linear function. Outputs: $\mathbb{R}^{64\times32\times32}$
4) *Average Pooling*: window size 1×2×2, stride (2,2). Outputs: $\mathbb{R}^{64\times16\times16}$
5) *Convolution*: window size 5×5 with stride (1,1), number of channels 64, pad (2,2), with bias. Outputs: $\mathbb{R}^{64\times16\times16}$
6) *ReLU*: ReLU non-linear function. Outputs: $\mathbb{R}^{64\times16\times16}$
7) *Average Pooling*: window size 1×2×2, stride (2,2). Outputs: $\mathbb{R}^{64\times8\times8}$
8) *Convolution*: window size 3×3 with stride (1,1), number of channels 64, bias is included for each channel. Outputs: $\mathbb{R}^{16\times8\times8}$
9) *ReLU*: ReLU non-linear function. Outputs: $\mathbb{R}^{64\times8\times8}$
10) *Convolution*: window size 1×1 with stride (1,1), number of channels 64, pad (1,1), with bias. Outputs: $\mathbb{R}^{64\times8\times8}$
11) *ReLU*: ReLU non-linear function. Outputs: $\mathbb{R}^{64\times8\times8}$
12) *Convolution*: window size 1×1 with stride (1,1), number of channels 16, bias is included for each channel. Outputs: $\mathbb{R}^{16\times8\times8}$
13) *ReLU*: ReLU non-linear function. Outputs: $\mathbb{R}^{16\times8\times8}$
14) *Fully Connected*: hidden neuron number 10. Outputs: $\mathbb{R}^{10\times1}$

Fig. 5: The neural network architecture for the CIFAR-10 dataset.

We compare Sphinx with the pure homomorphic encryption (pure-HE) training method, where the whole model is encrypted, and all homomorphic arithmetic operations occur between ciphertexts. Our pure-HE version of the training protocol is similar to [10]. For a fair comparison, we also adopt the client-aided method for non-linear layers in the pure-HE method to avoid expensive bootstrapping operations, but remove all the optimizations in Section VII. For both algorithms, we encrypt the input and bias weights with the lowest modulus levels available. We note that CryptoDL [22] also offers a HE-based privacy-preserving training. However, it focuses on how to accelerate the non-linear activation functions by replacing them with polynomial functions, while Sphinx makes no assumption on the model architecture and has no restrictions on the form of non-linear activation functions. Thus it is orthogonal to this paper.

Figure 6 and Figure 7 show the improvement of Sphinx over the pure-HE method in terms of training time and communication cost, respectively. We observe that Sphinx achieves about 35× less training time and 5× lower communication cost for both neural networks on MNIST and CIFAR-10 than the pure-HE method. The main reason behind such improvement is that Sphinx avoids most of the heavy homomorphic operations, including ciphertext-ciphertext matrix multiplication, rescaling, and relinearization, by protecting the linear components as plaintexts. Furthermore, with the fast homomorphic multiplication optimization, Sphinx enables a shallower ciphertext level with lesser computation and communication costs.

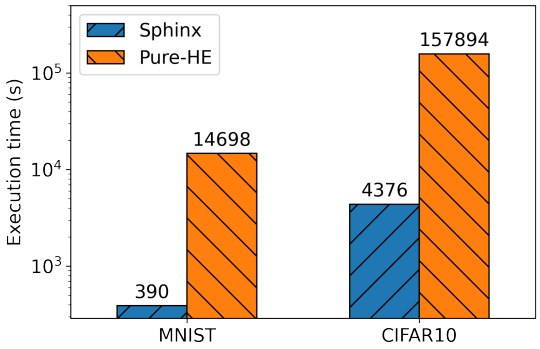


Fig. 6: The training execution times on Sphinx and pure-HE method for one batch.

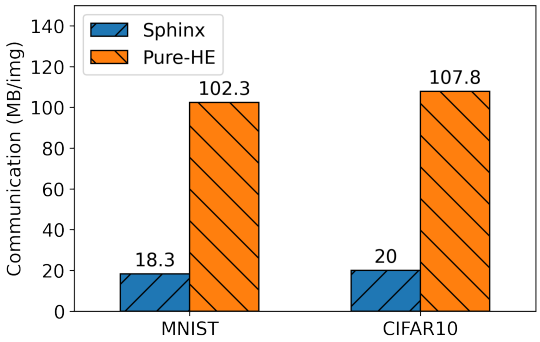


Fig. 7: The training communication costs on Sphinx and pure-HE method for one batch.

Table IV shows the breakdown performance in terms of the contribution by each optimization proposed in Section VII. We observe that while our naïve Sphinx achieves a 7-8× runtime improvement compared to pure-HE method, fast homomorphic multiplication optimization (Section VII-A) further makes another 4-5× improvement on it. However, fast homomorphic encryption optimization may cause a higher communication cost due to the larger ciphertext size with lazy relinearization. The other two optimizations, forward propagation cache and zero encryption/decryption (Section VII-B), reduce the communication overhead of the naïve method by about 20%, by reducing the required transferred ciphertexts and their sizes. Putting all optimizations together, Sphinx makes $5\times$ and $1.3\times$ improvements on the overall runtime and the communication cost, respectively, compared to the naïve method.

C. Inference with Sphinx

For the inference task, we conduct inference queries for a single sample and measure its inference latency. We compare our inference protocol with i) the pure-HE method mentioned above; ii) the forward propagation in the training protocol of Sphinx (Sphinx-FP), where noisy linear components, as well as all optimizations in Section VII, are used; iii) Gazelle [3], a hybrid private inference method based on HE and garbled circuits, and iv) Delphi [5], one of the state-of-the-art private inference protocols based on secret sharing and garbled circuits.

	Framework	Runtime (s)	Comm. (MB/image)
MNIST	Naïve Sphinx	1916	26.1
	+ FHM (§VII-A)	463	34.6
	+ FP-Cache (§VII-B)	1908	20.2
	+ ZE/ZD (§VII-B)	1890	22.5
	+ All	390	18.3
CIFAR-10	Naïve Sphinx	22182	28.8
	+ FHM (§VII-A)	4589	36.3
	+ FP-Cache (§VII-B)	22156	22.3
	+ ZE/ZD (§VII-B)	21937	23.7
	+ All	4376	20.0

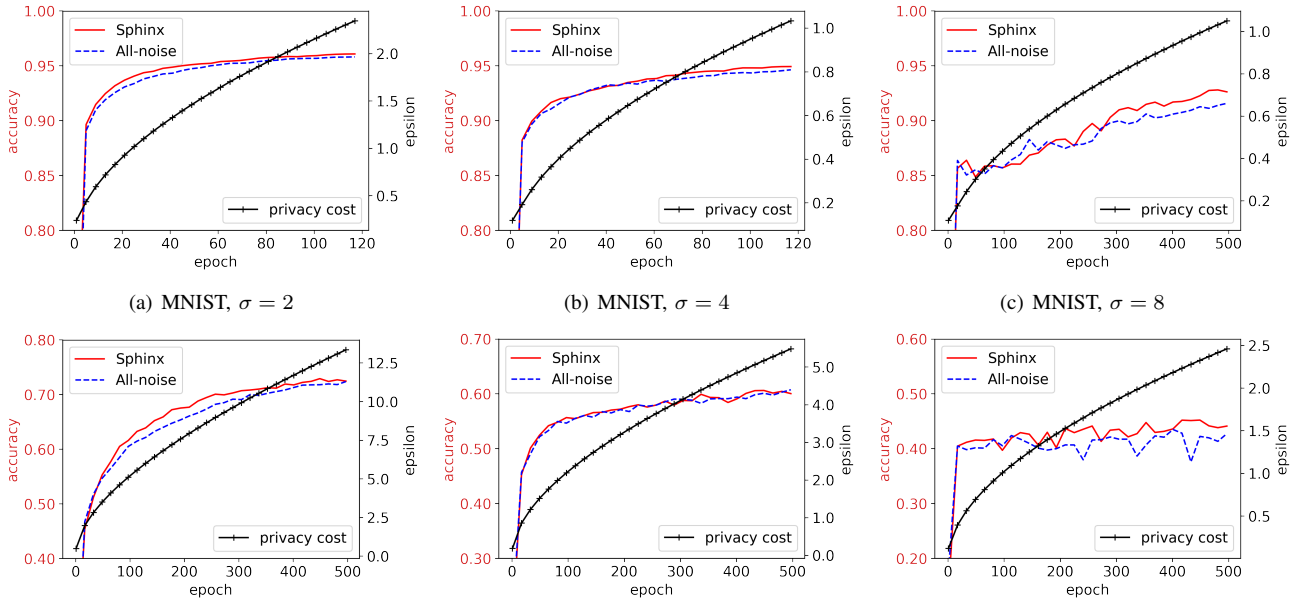
TABLE IV: The benchmarks of Sphinx variants for training one batch. Naïve Sphinx only uses plain noisy linear components. FHM means the fast homomorphic multiplication design. FP-Cache denotes the forward propagation cache. ZE/ZD means the zero encryption/decryption techniques.

	Framework	Time (s)		Communication (MB)	
		preproc.	online	preproc.	online
MNIST	Pure-HE	-	2915	-	13134
	Sphinx-FP	14.4	108	-	3014
	Gazelle	12.46	1.44	566.2	127.3
	Delphi	6.34	0.09	91.8	0.42
	Sphinx	6.01	**0.05**	87.6	**0.07**
CIFAR-10	Pure-HE	-	33909	-	53048
	Sphinx-FP	130	1065	-	13022
	Gazelle	42.65	4.82	1907	623
	Delphi	48.9	0.18	145	5.45
	Sphinx	48.3	**0.08**	128	**1.46**

TABLE V: The benchmarks of various inference protocols. Sphinx-FP prepares *zero ciphertexts* in the preprocessing phase. Delphi requires a plain public model on the server, thus cannot be used for privacy-preserving online learning.

Table V shows the latency and communication cost for inference with Sphinx in both preprocessing and online phases. Though we observe that the forward propagation of our training protocol in Sphinx already achieves almost $30\times$ faster inference latency and $4\times$ lower communication cost in both datasets compared to the pure-HE inference, they are still both expensive due to their homomorphic operations and batch-axis processing style[8]. However, our secret sharing based inference protocol in Sphinx achieves $5.8{*}10^4\times$ and $4.2{*}10^5\times$ faster inference latency and lower communication cost by about $1.9 * 10^5\times$ and $3.6 * 10^4\times$ in MNIST and CIFAR-10, respectively, compared to the pure-HE method, which is also comparable to state-of-the-art private inference methods such as Delphi. The reason behind the improvement is that Sphinx moves most of the heavy cryptographic operations to the preprocessing phase. Therefore, it executes the online inference involving only plain arithmetic operations over prime field, achieving significantly lower latency. Sphinx also achieves slightly better latency and communication overhead than Delphi because our setting avoids the overhead of heavy MPC techniques for non-linear layers, as mentioned in Section

[8]Sphinx-FP supports the batch size up to 2048 with the encryption parameters ($L = 1, N = 2^{12}$).



(a) MNIST, $\sigma = 2$ (b) MNIST, $\sigma = 4$ (c) MNIST, $\sigma = 8$

(d) CIFAR-10, $\sigma = 2$ (e) CIFAR-10, $\sigma = 4$ (f) CIFAR-10, $\sigma = 8$

Fig. 8: Results on the test accuracy and the privacy cost ϵ with Gaussian noise of different magnitudes on the MNIST and CIFAR-10 datasets. We fix the $\delta = 10^{-5}$ for all the experiments. One epoch denotes one iteration over the training dataset.

V-B. We note that these private inference protocols, including Gazelle and Delphi, require a public model and thus cannot be utilized in the online learning setting.

D. Privacy Cost of Sphinx

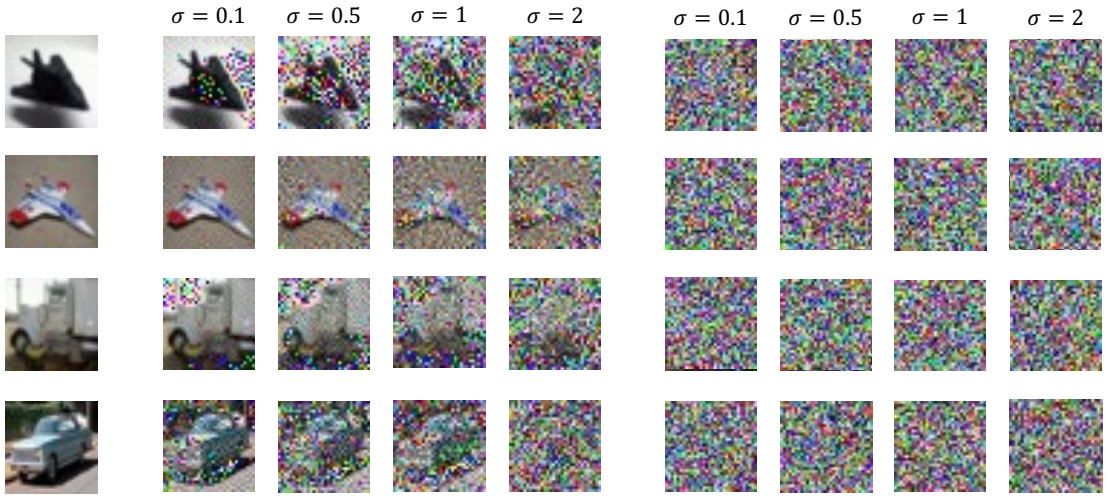
To evaluate the privacy-utility trade-off in Sphinx, we fix $\delta = 10^{-5}$ and keep track of privacy cost ϵ of the DP mechanism and the model performance during the training on MNIST and CIFAR-10, as suggested in [15]. To explore the influence of the noise mechanism on model performance, we conduct the experiments under different noise levels ($\sigma = 2, 4, 8$). The gradient norm bound C is set to 3. We compare our Sphinx with the *all-noisy algorithm*, where all model parameters, including the bias components, are protected with the Sampled Gaussian mechanism, thus stored in the server as plaintexts, which is equivalent to the DPSGD algorithm in [15].

Figure 8 shows the evolution of accuracy and privacy cost ϵ of Sphinx and the *all-noisy* algorithm as the training proceeds. We have several observations: i) Sphinx achieves comparable model accuracy compared with the all-noisy algorithm across various privacy budgets and noise magnitudes without a trusted data aggregator; ii) With a larger magnitude of noise (σ), the training process consumes the privacy budget more slowly (ϵ), yet leads to a more unstable learning process and harms the convergence rate. For example, when $\sigma = 8$ for MNIST and CIFAR-10, the model accuracies rise rapidly to 88% and 40% respectively in the first few epochs, but then struggle to climb slowly under strong fluctuations; iii) On MNIST, Sphinx achieves at most 92%, 94%, and 96% test accuracy for (0.5,10^{-5}), (1,10^{-5}), and (2,10^{-5})-DP under different noise levels. For CIFAR-10, Sphinx achieves 54%, 58%, and 72% test accuracy for (2,10^{-5}), (4,10^{-5}), and (12,10^{-5})-DP, which is slightly better across different noise levels compared to the *all-noisy* algorithm. It is because the gradients of the bias components are updated in encrypted form in Sphinx and thus exempted from the DP mechanism, resulting in overall lesser noise. Also, Sphinx only needs to clip the gradients over the norm of the linear components, which is smaller than that of the whole model: $\|\mathbf{g_W}\|_2 < \|\mathbf{g}\|_2$, resulting in lesser gradient loss during clipping and faster convergence.

We note that the accuracy drop of the differentially private models on CIFAR-10 is larger than those on MNIST. The phenomenon is also observed in previous work [15]. Besides the large data and task complexity of CIFAR-10, one reason is that deeper neural networks have more parameters, which leads to a larger $\|\mathbf{g_W}\|_2$. Thus, with the same norm bound C, more information in the gradients will be clipped. We have also tried to choose a larger C, which, however, will increase the added noise $N(0, \frac{\sigma^2 C^2}{B^2}\mathbf{I})$ given the same noise level σ and degrades the training performance even more. Due to the lack of a solid theoretical foundation of deep learning and poor interpretability of neural networks, how to choose a proper model architecture, noise level σ and norm bound C based on the task/dataset to maximize the model performance is a challenging topic, which we leave for future work.

E. Sphinx Against Attacks

To evaluate how well Sphinx defends against known attacks in practical scenarios, we conduct the *gradient-matching attack* proposed in [21] on the noisy gradients of Sphinx's linear components. The gradient-matching attack is a type of reconstruction attack method that can recover input images and labels from the gradients of the model. Initialized from



(a) Original (b) All noisy (c) Sphinx

Fig. 9: Visualization of the images recovered by the gradient-matching attack in [21] with different magnitudes of Gaussian noise: (a) the original images, (b) the images recovered from the gradients of all noisy model, where both the linear and bias components are plaintexts and protected by DP mechanisms, and (c) the images recovered from the gradients of Sphinx.

random input, it utilizes the gradient descent method to recover the input images and labels by minimizing the difference between the derived gradients and real gradients. We conduct the attack on both Sphinx and the *all-noisy* model under the CIFAR-10 configuration in Section IX-B. With a large batch size, the attack effectiveness will decrease significantly due to the huge number of possible combinations of individual gradients in a batch. Hence, we relax the constraint and feed the attacker with the individual gradients of input images, on which the introduced noise on the aggregated gradients is distributed uniformly. Because the gradient-matching attack requires second-order derivatives of the loss function, we replace the ReLU functions in non-linear layers with Sigmoid functions, which have also been widely used in various deep neural networks. For Sphinx, since the bias components are encrypted, the attacker cannot derive the simulated input gradients. Thus, we also initialize the bias components with random values, and the attacker needs to learn the input as well as the bias weights simultaneously.

Figure 9 shows the results of the attack. We observe that although the defense against gradient-matching attack strengthens with the increase in noise level, we can recover most pixels of the input image from the complete set of noisy gradients with small noise. With the noise level $\sigma = 2$, we can still recognize the objects in some of the recovered images. On the contrary, by masking the bias components with the HE scheme, which is only a small part of the entire model, Sphinx effectively defends the reconstruction attack than the pure DP algorithm, even when their privacy costs are almost the same as shown in Section IX-D. It is because the masking technique requires the attacker to infer the hiding model parameters and the input data simultaneously, which is often harder under the same noise level.

X. Conclusion and Future Work

In conclusion, we presented Sphinx, a privacy-preserving online learning system. Sphinx divides online learning into the training and inference phases and combines different privacy-preserving techniques reciprocally. Sphinx has significantly accelerated the privacy-preserving training and inference, enabling privacy-preserving online learning services on deep neural networks.

As Sphinx is compatible with any differentially private training algorithms that protect aggregated gradients in mini-batches, we can also replace the Sampled Gaussian Mechanism with other DP mechanisms with various features according to the application requirements and data characteristics. For example, Kairouz et al. proposed an online learning DP mechanism that allows a more flexible data access pattern [19], which can be adopted in Sphinx when the training data is Non-IID. We will explore and evaluate the combinations between different DP mechanisms and HE schemes in the future.

In the inference phase of Sphinx, we adopted the secret sharing technique in the preprocessing phase to accelerate the inference. However, we need to rerun the preprocessing phase upon the updated model parameters, causing a waste of computation resources. For future work, we envision a privacy-preserving online learning approach whose previous preprocessing results can be aligned with new model parameters with few extra calculations once the model is updated.

Acknowledgement

This work is supported in part by the Hong Kong RGC TRS T41-603/20R, GRF-16215119, GRF-16213621 and National Key R&D Program of China under Grant No.2018AAA0101100. We thank our shepherd and the anonymous reviewers for their constructive feedback and suggestions. Kai Chen is the corresponding author of this paper.

REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

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

[3] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1651–1669.

[4] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[5] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 2505–2522.

[6] J. L. Crawford, C. Gentry, S. Halevi, D. Platt, and V. Shoup, "Doing real work with fhe: the case of logistic regression," in *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2018, pp. 1–12.

[7] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang, "Secure logistic regression based on homomorphic encryption: Design and evaluation," *JMIR medical informatics*, vol. 6, no. 2, p. e19, 2018.

[8] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 334–348.

[9] S. Wang, Y. Zhang, W. Dai, K. Lauter, M. Kim, Y. Tang, H. Xiong, and X. Jiang, "Healer: homomorphic computation of exact logistic regression for secure rare disease variants analysis in gwas," *Bioinformatics*, vol. 32, no. 2, pp. 211–218, 2016.

[10] K. Nandakumar, N. Ratha, S. Pankanti, and S. Halevi, "Towards deep neural network training on encrypted data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.

[11] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[12] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 35–52.

[13] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 26–49, 2019.

[14] N. Agrawal, A. Shahin Shamsabadi, M. J. Kusner, and A. Gascón, "Quotient: two-party secure neural network training and prediction," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1231–1247.

[15] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[16] N. Phan, Y. Wang, X. Wu, and D. Dou, "Differential privacy preservation for deep auto-encoders: an application of human behavior prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[17] N. Phan, X. Wu, H. Hu, and D. Dou, "Adaptive laplace mechanism: Differential privacy preservation in deep learning," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 385–394.

[18] Z. Bu, J. Dong, Q. Long, and W. J. Su, "Deep learning with gaussian differential privacy," *Harvard data science review*, vol. 2020, no. 23, 2020.

[19] P. Kairouz, B. Mcmahan, S. Song, O. Thakkar, A. Thakurta, and Z. Xu, "Practical and private (deep) learning without sampling or shuffling," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 5213–5225. [Online]. Available: http://proceedings.mlr.press/v139/kairouz21b.html

[20] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1895–1912.

[21] B. Zhao, K. R. Mopuri, and H. Bilen, "idlg: Improved deep leakage from gradients," *arXiv preprint arXiv:2001.02610*, 2020.

[22] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service." *Proc. Priv. Enhancing Technol.*, vol. 2018, no. 3, pp. 123–142, 2018.

[23] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "Flash: Fast and robust framework for privacy-preserving machine learning." *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 2, pp. 459–480, 2020.

[24] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*. PMLR, 2016, pp. 201–210.

[25] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[26] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "ngraph-he2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2019, pp. 45–56.

[27] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "{XONN}: Xnor-based oblivious deep neural network inference," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1501–1518.

[28] A. Patra and A. Suresh, "Blaze: blazing fast privacy-preserving machine learning," *arXiv preprint arXiv:2005.09042*, 2020.

[29] A. Vizitiu, C. I. Nita, A. Puiu, C. Suciu, and L. M. Itu, "Applying deep neural networks over homomorphic encrypted medical data," *Computational and mathematical methods in medicine*, vol. 2020, 2020.

[30] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2006, pp. 486–503.

[31] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett, "Functional mechanism: regression analysis under differential privacy," *arXiv preprint arXiv:1208.0219*, 2012.

[32] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," *arXiv preprint arXiv:1610.05755*, 2016.

[33] L. Zhu and S. Han, "Deep leakage from gradients," in *Federated Learning*. Springer, 2020, pp. 17–31.

[34] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang, "Privacy loss in apple's implementation of differential privacy on macos 10.12," *arXiv preprint arXiv:1709.02753*, 2017.

[35] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 1054–1067.

[36] B. Hie, H. Cho, and B. Berger, "Realizing private and practical pharmacological collaboration," *Science*, vol. 362, no. 6412, pp. 347–350, 2018.

[37] R. Rachuri and A. Suresh, "Trident: Efficient 4pc framework for privacy preserving machine learning," 2022.

[38] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *Proceedings on Privacy Enhancing Technologies*, 2020.

[39] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.

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

[41] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for federated learning on user-held data," *arXiv preprint arXiv:1611.04482*, 2016.

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

[43] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[44] A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers, "Protection against reconstruction and its applications in private federated learning," *arXiv preprint arXiv:1812.00984*, 2018.
[45] B. Jayaraman and L. Wang, "Distributed learning without distress: Privacy-preserving empirical risk minimization," *Advances in Neural Information Processing Systems*, 2018.
[46] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, "Hybridalpha: An efficient approach for privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 13–23.
[47] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 1–11.
[48] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux, "Poseidon: Privacy-preserving federated neural network learning," 2022.
[49] Z. Liu, L. Wang, and K. Chen, "Secure efficient federated knn for recommendation systems," in *The International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*. Springer, 2020, pp. 1808–1819.
[50] D. Chai, L. Wang, K. Chen, and Q. Yang, "Secure federated matrix factorization," *IEEE Intelligent Systems*, 2020.
[51] L. Yang, B. Tan, V. W. Zheng, K. Chen, and Q. Yang, "Federated recommendation systems," in *Federated Learning*. Springer, 2020, pp. 225–239.
[52] D. Chai, L. Wang, L. Fu, J. Zhang, K. Chen, and Q. Yang, "Federated singular vector decomposition," *arXiv preprint arXiv:2105.08925*, 2021.
[53] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.
[54] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.
[55] "Microsoft SEAL (release 3.6)," https://github.com/Microsoft/SEAL, Nov. 2020, microsoft Research, Redmond, WA.
[56] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 395–412.
[57] S. Goryczka, L. Xiong, and V. Sunderam, "Secure multiparty aggregation with differential privacy: A comparative study," in *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, 2013, pp. 155–163.
[58] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients–how easy is it to break privacy in federated learning?" *arXiv preprint arXiv:2003.14053*, 2020.
[59] C. Dwork, G. N. Rothblum, and S. Vadhan, "Boosting and differential privacy," in *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE, 2010, pp. 51–60.
[60] I. Mironov, K. Talwar, and L. Zhang, "R\'enyi differential privacy of the sampled gaussian mechanism," *arXiv preprint arXiv:1908.10530*, 2019.
[61] M. Bun, C. Dwork, G. N. Rothblum, and T. Steinke, "Composable and versatile privacy via truncated cdp," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, 2018, pp. 74–86.
[62] M. Blatt, A. Gusev, Y. Polyakov, K. Rohloff, and V. Vaikuntanathan, "Optimized homomorphic encryption solution for secure genome-wide association studies," *BMC Medical Genomics*, vol. 13, no. 7, pp. 1–13, 2020.