PrivDNN: A Secure Multi-Party Computation Framework for Deep Learning using Partial DNN Encryption

Liangqin Ren
The University of Kansas
liangqinren@ku.edu

Zeyan Liu
The University of Kansas
zyliu@ku.edu

Fengjun Li
The University of Kansas
fli@ku.edu

Kaitai Liang
Delft University of Technology
kaitai.liang@tudelft.nl

Zhu Li
University of Missouri–Kansas City
lizhu@umkc.edu

Bo Luo
The University of Kansas
bluo@ku.edu

Abstract

In the past decade, we have witnessed an exponential growth of deep learning models, platforms, and applications. While existing DL applications and Machine Learning as a service (MLaaS) frameworks assume fully trusted models, the need for privacy-preserving DNN evaluation arises. In a secure multi-party computation scenario, both the model and the data are considered proprietary, i.e., the model owner does not want to reveal the highly valuable DL model to the user, while the user does not wish to disclose their private data samples either. Conventional privacy-preserving deep learning solutions ask the users to send encrypted samples to the model owners, who must handle the heavy lifting of ciphertext-domain computation with homomorphic encryption. In this paper, we present a novel solution, namely, PrivDNN, which (1) offloads the computation to the user side by sharing an encrypted deep learning model with them, (2) significantly improves the efficiency of DNN evaluation using partial DNN encryption, (3) ensures model accuracy and model privacy using a core neuron selection and encryption scheme. Experimental results show that PrivDNN reduces privacy-preserving DNN inference time and memory requirement by up to 97% while maintaining model performance and privacy. Codes can be found at https://github.com/LiangqinRen/PrivDNN

Keywords

Privacy-preserving Deep Learning, Homomorphic Encryption

1 Introduction

In the past decade, machine learning, especially deep learning [76], has empowered various applications, including image classification [53], object detection [83], and image segmentation [58], amongst others. Deep learning relies on complex models, such as deep neural networks (DNNs), that capture specific features from input data to perform these tasks. However, developing a large/complex, and highly accurate network requires enormous training data and computing resources. While smaller organizations, such as regional hospitals and local schools, cannot afford to train their models, they expect to use the advanced models the big companies provide as a service. When it applies to sensitive information such as public security [29], medical data [46, 73, 74], self-driving [60], manufacturing [77], or financial data [64], we face a privacy dilemma in that the user data must be revealed to the entity that evaluates with the model, or the model itself must be revealed to the user to evaluate locally. In these applications with sensitive data, we should protect data and DNN model privacy while making accurate predictions, which can be treated as a *secure two-party computation problem.*

Homomorphic encryption (HE) is a modern cryptographic technique based on lattices [3]. Fully homomorphic encryption (FHE) is the cryptosystem that supports addition and multiplication operations over encrypted data. Therefore, it is adopted in privacy-preserving outsourced storage and computation. Several works in past years introduced homomorphic encryption into deep neural networks, e.g., CryptoNets [26]. In privacy-preserving DNN model evaluation, users employ FHE to encrypt the data feed to the neural network so that the model owners perform inference over encrypted data, i.e., the plaintext raw data is hidden from the model owners. On the other hand, deep learning models trained from very large amounts of proprietary and potentially sensitive data also deserve protection. [4] is the first work that considers the protection of the model, and [43] is the first work that protects convolutional neural networks. They employ the same mechanism for data protection to protect the models, i.e., use FHE to encrypt the model and feed the encrypted model with plaintext data, which will be converted into the ciphertext domain during DNN evaluation. As a result, the evaluation result is encrypted with homomorphic encryption, and only the authorized user with the key (i.e., the model owner in this case) can decrypt it and recover the plain result.

However, the homomorphic operations are extremely inefficient compared with operations in the plaintext domain. Computational overhead became the main obstacle that limits its wide deployment. For example, CryptoNets requires 250 seconds to evaluate a small self-defined FHE-friendly model on encrypted samples from the MNIST dataset with an accuracy of 99%. Therefore, many works focus on improving the homomorphic encryption performance to make it more practical, such as [1, 7, 36].

In the applications where DNN models, instead of testing samples, are being protected, i.e., when the DNN models are shipped to the data owners to be executed locally, we do not necessarily have to encrypt the *entire* model. In this paper, we present the PrivDNN framework for DNN model protection, in which we encrypt a subset of the neurons/filters to ensure that the DNN would not provide a satisfactory performance without those protected neurons. In particular, the PrivDNN scheme identifies and protects

Proceedings on Privacy Enhancing Technologies 2024(3), 477–494

https://doi.org/10.56553/popets-2024-0089

(encrypts) those "important" attributes that significantly contribute to the performance of the target DNN. At the inference time, since the majority of the neurons that remain in the plaintext domain do not need to be evaluated with homomorphic operations, we expect to accelerate the evaluation speed dramatically. Moreover, we do not need to use the FHE-compatible activation functions for those plaintext attributes, which will also increase the model accuracy in comparison with the approaches that employ FHE-friendly activation functions for the entire model. In this paper, we present the design of the PrivDNN framework, with the algorithms for *core neuron selection*, i.e., to identify a small subset of neurons that ensures high performance for authorized DNN evaluation while significantly reducing DNN performance when it is evaluated without the contribution of the core neurons. Experiments with five popular benchmarking datasets and various DNN structures show that PrivDNN achieves the usability and security objectives while reducing private DNN evaluation time by order of magnitude compared to full-network-encryption approaches. Finally, we would like to highlight that the key idea of PrivDNN, to reduce the FHE operations in DNN evaluation through partial DNN encryption, does not rely on any specific homomorphic encryption algorithms, homomorphic operations, or DNN architecture. Therefore, it could be adopted in any FHE-enabled DNN model encryption scheme to significantly improve its efficiency. In summary, the main contributions of this paper are three-fold.

(1) We articulate the different models for privacy-preserving DNN evaluation and elaborate on the necessity for the model encryption approaches.
(2) We present the first practical system PrivDNN that protects deep neural network models with homomorphic encryption, which achieves sufficient protection while dramatically reducing the computation to enable efficient privacy-preserving DNN evaluation over large DNN models.
(3) We design and compare three different schemes for core neuron selection. We further demonstrate the effectiveness of the PrivDNN approach through extensive experiments.

The rest of the paper is organized as follows: We introduce the background and preliminaries in Section 2, followed by the formal problem statement and threat model in Section 3. We present the technical details of PrivDNN, the experimental results, and the security analysis in Sections 4, 5, and 6. We summarize the literature in Section 7 and finally conclude the paper in Section 8.

2 Preliminaries

SMC and Homomorphic Encryption. Secure multiparty computation (SMC) enables mutually untrusted parties to jointly perform computing tasks without disclosing their private inputs [21]. SMC was first introduced by Yao in The Millionaires' problem [79]. More recently, it has been introduced to new applications such as secure cloud computing and secure machine learning [82].

Homomorphic encryption (HE) allows users to perform computations on encrypted data without first decrypting it. It has been adopted in SMC applications, including privacy-preserving machine learning (PPML) [2]. HE schemes are roughly categorized into: *partial homomorphic encryption*, *somewhat or leveled homomorphic encryption*, and *fully homomorphic encryption* (FHE). FHE [25, 69] supports unlimited additions and multiplications in the ciphertext domain but requires excessive computation. There are three popular FHE schemes: *Brakerski/Fan-Vercauteren (BFV)* [24], *Brakerski-Gentry-Vaikuntanathan (BGV)* [6] and *Cheon, Kim, Kim and Song (CKKS)* [12]. BFV and BGV support accurate calculations over integers, while CKKS supports calculations over floats with errors.

Deep Learning. A typical DNN includes an input, output, and several hidden layers, such as:
(1) The convolution layer multiplies the input by a vector of weights and sums the results. The weights are calculated during training. The operations in the convolution layers are supported in FHE.
(2) The pooling layer performs downsampling by dividing the input into pooling regions and computing each region's maximum or average value. Since homomorphic encryption only supports addition and multiplication operations, the FHE-supported DNNs always use average pooling instead of maximum pooling.
(3) The activation layer contains nonlinear functions to enable DNNs to learn complex patterns. The common activation functions include Sigmoid [30], Tanh [45], ReLU [27], and Swish [67]. To implement non-linear activation in FHE, we can use polynomial substitution [26], precompute function for discrete values [5, 15], use low-degree polynomials to approximate the non-linear functions [9, 13, 36, 61], or implement nonlinear functions through sign function [51].
(4) The batch normal (BN) layer reduces internal covariate shift by performing the normalization for each training mini-batch to allow higher learning rates. The BN layer supports FHE by using the average and variance calculated in the training process [41].

A large DL model may have millions of parameters that require excessive computing power, memory, and storage [19]. Network pruning [50] has been identified as an effective technique to improve efficiency. A typical pruning process includes three steps: train a large over-parameterized model, prune the model, and fine-tune the pruned model to regain the lost performance [35]. Pruning can be categorized into *unstructured pruning* [31], which removes specific weights of neural networks to achieve a high compression rate, and *structured pruning* [52] pruning, which removes entire filters to accelerate with standard hardware and libraries.

3 Problem Statement and Attack Model

3.1 Background and SMC Models for Privacy-preserving DNN Evaluation

We assume a typical client-server scenario for DNN evaluation. The Server, $\mathcal{S}$, is the owner of a proprietary DNN, which is trained on private data belonging to $\mathcal{S}$. The Client, C, is the user who has private data samples that need to be evaluated by $\mathcal{S}$'s DNN. In the conventional client-server DNN evaluation scenario, as shown in Figure 1 (A), C ships her raw data to $\mathcal{S}$, who performs inference in cleartext. While this approach is the most efficient, it reveals the client's data to the server, which could be undesirable or impossible in specific applications due to business, ethical, or legal considerations. For instance, consider a small regional hospital that wants to employ deep learning to predict patients' cancer categories to prepare for subsequent treatment. Due to its limited samples, it is impractical for the hospital to train its own DL model. The hospital may employ high-performance models trained by large organizations, such as Merative. However, the hospital may not be

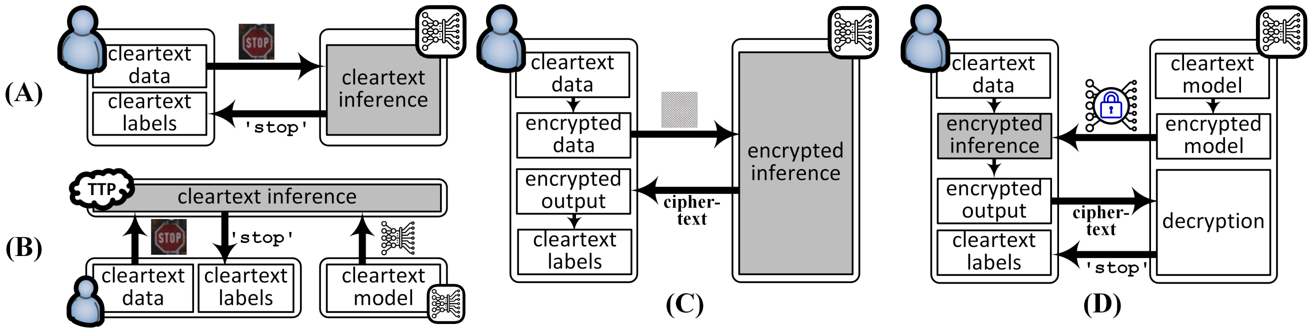


Figure 1: Approaches for privacy-preserving DNN evaluation: (A) clear text approach (no privacy protection); (B) trusted third party approach; (C) data encryption approach; and (D) model encryption approach.

able to ship raw data samples to the service provider due to legal requirements regarding the confidentiality of the patient, such as HIPAA [22]. Meanwhile, the owner of the large DL models is unwilling to freely share the trained model with the community due to its business interests: the models are trained with large amounts of proprietary data and extensive computing resources, both of which could be highly valuable. In this scenario, a secure multi-party computing (SMC) mechanism is desired, and two types of privacy are considered: *DNN model privacy* and *data privacy*.

• **DNN Model Privacy.** From the model owners' perspective, they would like to maintain the ownership of the high-performance DL model. That is, the users should not be able to replicate or use the model without the owners' permission.

• **Data Privacy.** From the users' perspective, they will not reveal the raw data to the model owners or any (untrusted) third parties.

As shown in Figure 1, there exist four approaches for the client-server DNN evaluation problem. Here, we briefly describe each approach and discuss its advantages and disadvantages.

1. The Clear Text Approach. As shown in Figure 1 (A), C sends the plaintext data to $\mathcal{S}$, who performs inference in plaintext. *Advantages*: This approach preserves DNN model privacy since the model never leaves its owner. It is very efficient since all computations are in plaintext. *Disadvantages*: This approach does not preserve the users' data privacy, i.e., all data samples are revealed to $\mathcal{S}$. An alternate approach is to send the model to C to perform inference, which preserves the user's data but not the owner's DNN model.

2. The Trusted Third-Party Approach. As shown in Figure 1 (B), instead of having $\mathcal{S}$ and C trusting each other, they identify a third party that is trusted by both of them. The DNN model and the data are shipped in plaintext to the third party, who performs inference. *Advantages*: This approach is easy to implement and fast since the inference phase is performed in plaintext. *Disadvantages*: Identifying a trusted third party could be challenging or even impractical due to legal requirements.

3. The Data Encryption Approach. Privacy-preserving DNN evaluation mechanisms have been proposed to perform inference on encrypted data [26]. As shown in Figure 1 (C), C encrypts her testing samples using homomorphic encryption and ships the encrypted data to $\mathcal{S}$, who performs inference in the ciphertext. The model output is returned to the user in ciphertext, who decrypts the data to retrieve the label. *Advantage:* $\mathcal{S}$ and C do not trust each other while both model privacy and data privacy are preserved. Privacy protection is enforced by cryptographic properties rather than trust. *Disadvantage:* The computation overhead is significant since the inference is performed in the ciphertext. Moreover, all the excessive ciphertext computations are performed by $\mathcal{S}$, who may not have the incentive to do so since the primary beneficiary is C.

4. The Model Encryption Approach. As shown in Figure 1 (D), $\mathcal{S}$ encrypts the plaintext model with homomorphic encryption and sends the encrypted model to C, who performs model evaluation in the ciphertext domain. The encrypted model output is returned to $\mathcal{S}$, which decrypts the data and returns the cleartext label. *Advantages:* Both DNN model privacy and data privacy are perfectly preserved through cryptography (like the data encryption approach). The heavy-lifting computation is moved to C, who should be willing to handle the overhead while enjoying the benefit of data privacy. *Disadvantages:* The computation overhead is significant. Compared with the data encryption approach, an additional one-way network transmission is required for each inference.

3.2 The Threat Model

We adopt the *model encryption approach* (Figure 1 (D)) for privacy-preserving DNN evaluation. Our scenario involves two parties: the model owner (server $\mathcal{S}$) and the data owner (client C). We assume all participants to follow the honest-but-curious (semi-honest) model [20, 28], i.e., they precisely follow the protocol (honest), while they also actively attempt to obtain or infer knowledge about others (curious). In particular, we make the following assumptions:

1. The data owner C knows the architecture of the model and a portion of the parameters transmitted to her in plaintext.

2. The data owner cannot break the encryption to learn the encrypted parameters of the DNN model. We employ a SOTA FHE scheme, CKKS [12], to encrypt DNN parameters. We assume that both the CKKS scheme and its implementation are secure. That is, only the model owner $\mathcal{S}$ is capable of decrypting any ciphertext.

3. The data owner does not have $\mathcal{S}$'s proprietary data used to train the model. Otherwise, she could train her model from scratch.

4. The security of the computing platforms and the communication channels are considered outside of the scope of this work.

Finally, the PrivDNN framework is designed to be adaptable for all models and FHE-encryption algorithms. That is, its functionality does not rely on any specific feature of a deep learning model, e.g., it does not assume the existence of a certain type of neurons, nor does it prohibit any neurons in the target DNN model.

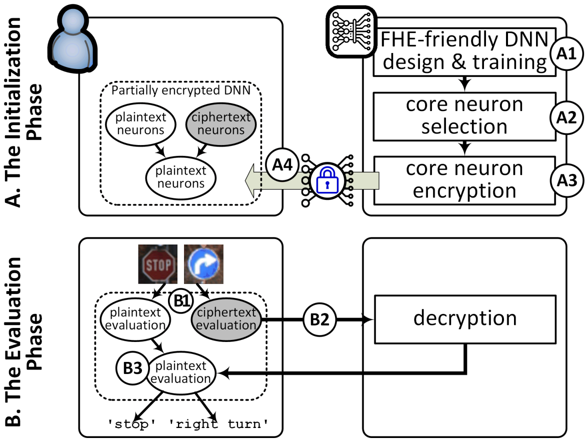


Figure 2: An overview of the PrivDNN approach.

4 The PrivDNN Approach

4.1 PrivDNN Overview

An overview of the proposed PrivDNN approach is shown in Figure 2. As we have described in the system model, the privacy-preserving DNN evaluation system involves two parties: the model owner ($\mathcal{S}$) and the data owner ($\mathcal{C}$). PrivDNN contains two stages: the initialization phase and the evaluation phase.

The *initialization phase* is a one-time process, which includes model training and protection: (A1) The model owner $\mathcal{S}$ designs an FHE-friendly DNN and trains it with her proprietary data. The model owner may also convert a pre-trained DNN to an FHE-friendly version with a small performance penalty. (A2) $\mathcal{S}$ identifies a set of *core neurons* that are essential to the performance of the network (to be articulated in Section 4.4). (A3) $\mathcal{S}$ employs a homomorphic encryption scheme to encrypt the core neurons. Finally, (A4) the partially encrypted DNN, which contains both plaintext and ciphertext neurons, is sent to the data owner $\mathcal{C}$.

The *evaluation phase* is invoked each time a batch of testing samples is evaluated by the private DNN: (B1) The data owner $\mathcal{C}$ evaluates the testing samples through the partially encrypted layers. Evaluation with the unencrypted neurons is performed in plaintext. In contrast, computation with the encrypted core neurons is performed in ciphertext using FHE. (B2) The output from the encrypted neurons will be obfuscated (multiplied by a random value), shipped to $\mathcal{S}$ for decryption, sent back to $\mathcal{C}$, and deobfuscated. (B3) With the output from the plaintext neurons and the decrypted output from the core neurons, $\mathcal{C}$ continues to evaluate the testing sample with the remaining plaintext layers of the DNN, to obtain the final output, i.e., the label for the testing sample.

In this section, we describe each component in both the initialization and the evaluation phases and articulate the technical details of the core neuron selection and partial DNN encryption mechanisms.

Notations. A pre-trained DNN model has a set of L convolutional layers and N neurons, among which L^i is the i-th convolutional layer and N_e is the number of encrypted neurons/filters. The model accuracy is A_o. The parameters in L^i can be represented as a set of 4-D filters[1] $W_{L^i} = \{W_1^i, W_2^i, ..., W_{n_i}^i\} \in \mathbb{R}^{n_{i-1}\times n_i\times k_i\times k_j}$, where the j-th filter is $w_j^i \in \mathbb{R}^{n_{i-1}\times k_i\times k_j}$. n_i is the number of filters in L_i. $k_i\times k_j$ represents the kernel size (usually k_i=k_j). The output of the filters, i.e., feature maps, are denoted as $O^i = \{o_1^i, o_2^i, ..., o_{n_i}^i\}$, where the j-th feature map $o_j^i \in \mathbb{R}^{c\times h_i\times w_j}$ is generated by w_j^i. c is the channel of input images. h_i and w_i are the height and width of the feature map. In PrivDNN, we propose to encrypt a subset of the filters in the model to accelerate the ciphertext evaluation. Therefore, W_{L^i} could be split into two groups, i.e., a subset to be encrypted $E_{L^i} = \{w_{E_1^i}^i, w_{E_2^i}^i, ..., w_{E_{n_{i1}}^i}^i\}$ and a subset in plaintext $P_{L^i} = \{w_{P_1^i}^i, w_{P_2^i}^i, ..., w_{P_{n_{i2}}^i}^i\}$, where E_j^i and P_j^i together represent the j-th filter.

4.2 FHE-Friendly DNN Design

The inference of PrivDNN requires computation in the ciphertext domain. Therefore, the operations involved in the encrypted neurons must be FHE-friendly. Existing works on FHE-friendly activation functions could be roughly grouped into three categories: (1) using a polynomial substitution [26], (2) precomputing function for discrete values [15], and (3) using low-degree polynomials to approximate the non-linear functions[13, 42, 61].

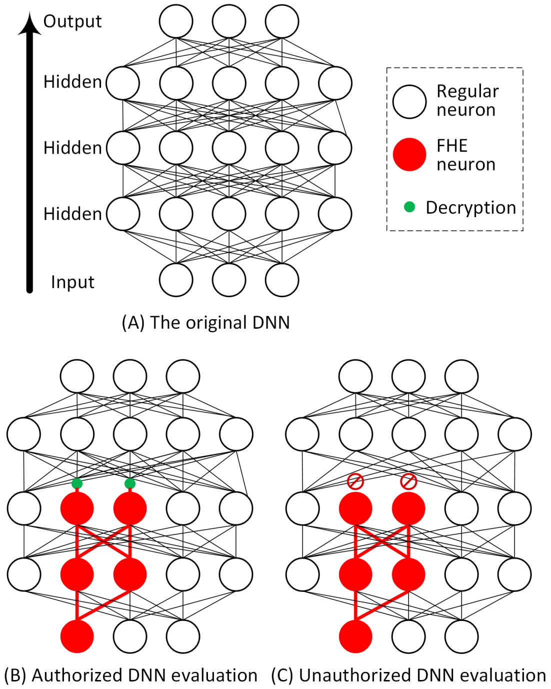
To partially mitigate the requirement of FHE-friendly activation functions, PrivDNN only requires the subset of neurons selected to be encrypted to be FHE-friendly. For example, when only the neurons from the first two convolutional layers are selected as the core neurons, we only need one linear activation function for the pooling layer because the data owner could decrypt the output from the second convolution layer.

4.3 Partial DNN Encryption: Motivation and Objectives

In PrivDNN, our core idea is to select a subset of *core neurons* to be protected (encrypted) so that: (1) the accuracy of the deep neural network will decrease significantly if the core neurons are removed from the network, (2) only a small set of neurons are selected as core neurons so that the majority of the computation for DNN evaluation is still in plaintext. Meanwhile, since the output of an encrypted neuron will be ciphertext, we cannot feed it into a plaintext neuron unless it is decrypted. Otherwise, the neuron must be converted to ciphertext for FHE computation. Therefore, PrivDNN generates a partially encrypted DNN with the core neurons in the ciphertext world and the other neurons in the plaintext world.

An example of the PrivDNN approach is shown in Figure 3. Figure 3 (A) demonstrates an original deep neural network, whose accuracy is denoted as A_o. As shown in Figure 3 (B) and (C), we consider two operation modes for a partially encrypted DNN:
(1) **Authorized DNN evaluation**. The FHE-encrypted neurons take inputs from both unencrypted neurons and other FHE-encrypted neurons. Inputs from unencrypted neurons will be encrypted to participate in cipher-domain computations. Since the output of FHE-encrypted neurons is all in the ciphertext domain, they cannot serve as input to unencrypted neurons. Therefore, as shown in Figure 3 (B), the execution of the DNN is naturally *separated* into the cleartext world (white) and the ciphertext world (red). Computation in the cleartext world is the same as regular DNN evaluation, except that it does not take input from the ciphertext world. FHE supports computation in the ciphertext world. It takes input from both the

[1] In this paper, we use *filters* and convolution layer *neurons* interchangeably.



(A) The original DNN

(B) Authorized DNN evaluation (C) Unauthorized DNN evaluation

Figure 3: The partial DNN encryption approach.

ciphertext world and the cleartext world, but it does not generate output to the cleartext world until the very last layer, where the ciphertext outputs are decrypted by $\mathcal{S}$ (shown in green in Figure 3 (B)). We denote the accuracy of authorized DNN evaluation as A_s. Since the connections from FHE neurons to unencrypted neurons are eliminated, we expect A_s to be slightly lower than A_o.

(2) **Unauthorized DNN evaluation**. As shown in Figure 3 (C), an unauthorized user who obtained the partially encrypted DNN cannot decrypt the final output from the ciphertext world and join it with the plaintext world. Therefore, the FHE neurons are practically *removed* from the DNN. We denote the testing accuracy of the unauthorized execution as A_r. Again, A_r is expected to be much lower than A_o due to the elimination of the neurons.

Therefore, the primary challenge of the PrivDNN framework is to designate a subset of neurons in the target DNN as the *core neurons* so that the following design objectives are satisfied:

G1. The Usability Goal: Maximize the Authorized Evaluation Accuracy. The partially encrypted DNN should maintain high accuracy so that it is usable to authorized clients, i.e., the accuracy for authorized users, A_s, should be very close to the accuracy of the original model A_o. That is, $A_o - A_s$ should be close to 0.

G2. The Security Goal: Minimize the Unauthorized Evaluation Accuracy. The partially encrypted network is well protected against unauthorized users so that the model owner's valuable information is preserved. The model accuracy without the protected core neurons, A_r, should be significantly lower than the authorized evaluation accuracy A_s, i.e., $A_s - A_r$ should be large.

G3. The Performance Goal: Minimize the Number of FHE-Encrypted Neurons. The ciphertext operations are significantly slower than plaintext operations. Hence, we want to identify a relatively smaller set of core neurons so as to reduce the amount of ciphertext operations. For instance, for a CNN with N convolution neurons (filters), if we identify N_e filters to be encrypted, the computation of the partially encrypted model is *roughly* N_e/N of a full-model-encryption solution. In PrivDNN, we define N_e as a computation budget preset by the model owner. If the model owner has an estimation of the cost of full model encryption and chooses to accept 10% of its cost, she would designate $N_e \simeq 0.1N$ (please see Section 5.3 for more discussions on computation).

With a preset N_e, core neuron selection aims to maximize A_s and minimize A_r simultaneously. To evaluate whether/how a selection scheme satisfies the design objectives, we define the *core set quality score* s to measure the quality of the selected core neuron set:

$$s = -sigmoid(A_o - A_s) + sigmoid(\frac{A_s - A_r}{\alpha}) \tag{1}$$

The first component denotes a *penalty* when the model accuracy decreases for authorized DNN evaluation, where $-1 < A_o - A_s < A_o < 1$. We pick the sigmoid function ($y = 1/(1 + e^{-x})$) to enforce a substantial penalty even for a small value of $A_o - A_s$. The second component denotes a *benefit* when the model accuracy decreases for unauthorized evaluation. A higher score s indicates a high-quality selection of the core neurons, i.e., a small $A_o - A_s$ for **G1** and a large $A_s - A_r$ for **G2**. In contrast, a negative s indicates that the model performs badly for authorized users (large $A_o - A_s$ that outweighs the benefit of model protection). Since $A_o - A_s$ and $A_s - A_r$ have different ranges, we use a coefficient α as a normalization factor. In the experiments, we use an empirically selected value $\alpha = 5$.

4.4 Core Neuron Selection in PrivDNN

We present the following four algorithms for core neuron selection. In all following algorithms, we assume that the core neurons are selected from l consecutive convolutional layers, where n_{i1} neurons are to be selected at layer i, while n_{i2} neurons at layer i remain in the cleartext domain. n_i denotes the total number of neurons at this layer so that $n_i = n_{i1} + n_{i2}$. Note that n_{i1} and n_{i2} are pre-selected by the model owner $\mathcal{S}$ to balance the trade-off between model security and efficiency (please see Section 6.4 for more discussion).

For a target DNN model with N convolution neurons, an exhaustive search of N_e core neurons will involve testing A_s and A_r on $\binom{N}{N_e}$ different settings. While this may be feasible for small DNNs, it is impractical for any complex network. Therefore, we introduce the random selection approach as the baseline.

Baseline: Random Selection. Starting from a pre-selected layer, e.g., Layer 1, we randomly identify n_{i1} neurons from layer i as core neurons. The algorithm is presented in Algorithm 1. The random approach does not optimize the selection process toward the selection objectives. We introduce it as a reference to demonstrate the optimization performance of the next three approaches.

The Greedy Approach. We aim to select core neurons that maximize s as defined in Equation (1). We design a greedy algorithm (Algorithm 2) to select the neurons layer-by-layer. In layer i, we first temporarily add each neuron j to the core neuron set (denoted as E_{Li}) and then evaluate the DNN performance A_r and A_s to calculate the core set quality score s_j. We select the neuron that produces the best s_j and permanently add it to core neurons. We repeat the process for this layer until n_{i1} neurons are added to the core set. We then move to the next layer until we finish all l layers.

Algorithm 1: The Algorithm for Random Selection

```
Data: well-trained model $M$ with $L$ layers
Result: selected core neuron set $E_L$
$i \leftarrow GetFirstLayerIndex(M)$;
$l \leftarrow GetSelectLayerCount(M)$;
for $i \leftarrow i$ to $i + l$ do
    $n_{i1} \leftarrow GetSelectFilterCount(M, i)$ ;
    $n_i \leftarrow GetFilterCount(M, i)$ ;
    $E_{L^i} \leftarrow RandomSelectFilters(n_i, n_{i1})$ ;
end
```

Algorithm 2: The Algorithm for Greedy Selection

```
Data: well-trained model $M$ with $L$ layers
Result: selected core neuron set $E_L$
$i \leftarrow GetFirstLayerIndex(M)$;
$l \leftarrow GetSelectLayerCount(M)$;
for $i \leftarrow i$ to $i + l$ do
    $n_{i1} \leftarrow GetSelectNeuronCount(M, i)$ ;
    $P_{L^i} \leftarrow GetAllNeurons(M, i)$ ;
    $E_{L^i} \leftarrow \{\}$ ;
    while $count(E_{L^i}) < n_{i1}$ do
        foreach $f$ in $P_{L^i}$ do
            $s[f] \leftarrow CalculateQuality(M, E_L, f)$ ;
        end
        $m \leftarrow argmax(s[f])$ ; /* "best" (one) neuron */
        if $s[m] \leq s[E_{L^i}[-1]]$ then
            break ;                          /* optional */
        end
        $E_{L^i} \leftarrow E_{L^i} \cup \{m\}$ ;
        $P_{L^i} \leftarrow P_{L^i} - \{m\}$ ;
    end
end
```

The greedy selection is a dynamic evaluation approach in which the accuracy of the target DNN (both A_s and A_r) is evaluated for different core neuron selections. Therefore, the greedy approach is computationally more expensive than static analysis, in which the core neurons are selected by analyzing the architecture and weights of the target neural network. Moreover, the greedy selection may reach a local but not the global optimum. In the experiments, we show that the greedy approach slightly compromises selection efficiency to achieve a high quality of the core neuron set.

The Pruning-based Approach. Research on DNN pruning aims to remove redundant filters in a DNN to produce sparse models for performance acceleration and model compression. The neurons/filters left after the pruning are supposed to be the ones that contribute the most to the model's performance. Therefore, our application may adopt the pruning algorithms in the literature for core neuron selection. As shown in Algorithm 3, we assume that an arbitrary pruning function $Prune(M, i, n_{i2})$ is employed to identify n_{i2} neurons in the ith layer of DNN model M to be pruned. The remaining n_{i1} unselected neurons are added to the core neuron set. Note that we do not perform the actual pruning operation, i.e., the "pruned" neurons are left intact in the model.

Algorithm 3: The Algorithm for Pruning-based Selection

```
Data: well-trained model $M$ with $L$ layers
Result: selected core neuron set $E_L$
$i \leftarrow GetFirstLayerIndex(M)$;
$l \leftarrow GetSelectLayerCount(M)$;
for $i \leftarrow i$ to $i + l$ do
    $n_{i1} \leftarrow GetSelectFilterCount(M, i)$ ;
    $n_i \leftarrow GetFilterCount(M, i)$ ;
    $P_{L^i} \leftarrow GetAllNeurons(M, i)$ ;
    $E_{L^i} \leftarrow P_{L^i} - Prune(M, i, n_i - n_{i1})$ ;
end
```

PrivDNN requires the following for the pruning algorithm: (1) The algorithm should be structured pruning. Limited by FHE properties, our protection is based on the neurons (filters) rather than weights. Therefore, we can only employ structured pruning, not weight pruning. (2) The algorithm must prune a pre-trained model instead of establishing a new, smaller model from scratch [40, 56]. (3) The algorithm should be one-shot rather than progressive [8, 70] because we cannot modify the model. (4) The algorithm should generate a static rather than a dynamic selection that changes based on the input. In PrivDNN, the model owner $\mathcal{S}$ will share a partially encrypted model with the data owner C. Since the model owner does not know any information about the dataset at the downstream user C, we must use algorithms that can generate static selections rather than the dynamic pruning algorithms that change the selection based on the testing data [33, 54].

Considering the above four limitations, we select four classic pruning algorithms: PFEC [52], FPGM [34], HRank [55], and GFS [80]. The first two are weight-dependent algorithms, and the other two are not. PFEC[52] proposes to prune filters based on their l-norm values. When a filter with smaller l-norm values multiplies the input, its output is also more likely to be smaller, i.e., its output is more likely to fail to pass sectional activation functions like ReLU. It does not contribute to the next layer. Therefore, there is unlikely any harm in pruning this filter. FPGM [34] argues that the norm-based criterion may not be accurate when the norm deviation is too small or the minimum norm of filters is not small. Therefore, it prunes filters with redundant information, i.e., the filters can be replaced with other filters. HRank [55] suggests that filters with a higher rank should have more information. At the same time, the rank of a specific filter is relatively stable with different inputs. Therefore, HRank calculates filters' ranks with a small data set, and prunes filter with smaller ranks. GFS [80] is a forward greedy algorithm. Most pruning algorithms will start with the complete model and remove filters based on their strategies. GFS removes all filters at the beginning and adds filters back individually. During this process, GFS calculates the loss that adding back a filter will decrease and selects the filter that decreases the most loss.

The Pruning+Greedy Approach. Both pruning and greedy strategies have advantages and disadvantages. The greedy algorithm provides solid selection performance (will elaborate in Section 5), but it is less efficient since it requires on-the-fly evaluation of DNN models in the selection process. The static-analysis-based pruning approaches are very fast, however, they select a subset of neurons that do not provide optimal accuracy for authorized users.

Algorithm 4: Pruning+Greedy Selection

```
Data: well-trained model $M$ with $L$ layers
Result: selected core neuron set $E_L$
$i \leftarrow GetFirstLayerIndex(M)$;
$l \leftarrow GetSelectLayerCount(M)$;
for $i \leftarrow i$ to $i + l$ do
    $n_{i1} \leftarrow GetSelectFilterCount(M, i)$ ;
    $n_i \leftarrow GetFilterCount(M, i)$ ;
    $P_{L^i} \leftarrow GetAllNeurons(M, i)$ ;
    while $count(E_{L^i}) < n_{i1}$ do
        foreach $f$ in $P_{L^i} - Prune(M, i, (n_i - n_{i1}) \times p)$ do
            $s[f] \leftarrow CalculateQuality(M, E_L, f)$ ;
        end
        $m \leftarrow argmax(s[f])$ ; /* “best” (one) neuron */
        if $s[m] \leq s[E_{L^i}[-1]]$ then
            break ;                        /* optional */
        end
        $E_{L^i} \leftarrow E_{L^i} \cup \{m\}$ ;
    end
end
```

Table 1: Datasets and models used in the experiments.

	M	E	G	C	T
Train size	60000	124800	39209	50000	100000
Test size	5000	10400	6315	5000	10000
DNN Model	LeNet-5	LeNet-5	AlexNet	VGG16	ResNet18
Accuracy(A_o)	99.36%	93.08%	93.51%	90.22%	72.00%

1. M: MNIST [18], E: EMNIST [14], G: GTSRB [38], C: CIFAR10 [47], T: Tiny-ImageNet [59] **2.** We adopt top-5 accuracy for Tiny-ImageNet.

Therefore, we propose integrating these two strategies as shown in Algorithm 4. We first employ the pruning algorithm to exclude the clearly redundant neurons, i.e., to reduce the size of the selection pool for the subsequent greedy algorithm. In practice, we create a greedy selection pool whose size is p times the target core set size (we empirically set $p = 2$ in our experiments). Then, we use the greedy strategy to select filters from the pool.

5 Experiments

5.1 Settings

We implement the PrivDNN framework with all three core neuron selection approaches (presented in Section 4.4) using Python 3.10.13, PyTorch 2.1.0, and CUDA 12.1. All the experiments presented in this section are performed on a desktop computer with Ubuntu 22.04 LTS running on AMD Ryzen 7 3700X eight-core CPU, NVIDIA 3090 GPU, and 64 GB memory. To support DNN evaluation in the ciphertext domain, we adopt the CKKS [12] scheme as implemented in Microsoft SEAL 4.1.1 [72] library. As shown in Table 1, we adopt five popular benchmarking datasets (See Appendix C for details).

As discussed in Section 4.2, PrivDNN, as well as all FHE-based privacy-preserving DNN evaluation schemes in the literature, requires FHE-friendly DNN models. We use the FHE-friendly square function as the ciphertext-domain activation function. In practice, PrivDNN may train the FHE-friendly DNN from scratch or protect a pre-trained DNN. We convert the activations to FHE-friendly for pre-trained models and tune the modified model. In the experiments, we train all the models from scratch. As shown in Table 1, the FHE-friendly models for MNIST, EMNIST, GTSRB, CIFAR10, and Tiny-ImageNet get the accuracy (A_o) of 99.36%, 93.08%, 93.51%, 90.22%, and 72.00%. We focus on protecting the model rather than increasing the model's training performance or accuracy. Hence, we use the same parameters for the scheduler, batch size (128), and epochs (128) in all models.

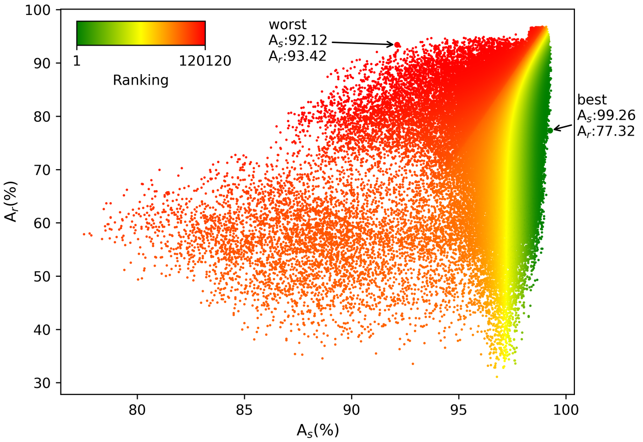


Figure 4: The distribution of $\{A_s, A_r\}$ for all possible core neuron set selections for MNIST/LeNet-5 ($n_{1,1} = 2, n_{2,1} = 6$).

5.2 Core Neuron Selection

Ground Truth. As discussed in Section 4.4, we should perform an exhaustive search to learn the quality of all possible core neuron sets. For instance, to encrypt n_{i1} neurons out of n_i neurons from layer i, there are $\binom{n_i}{n_{i1}}$ different selections to be evaluated. Assume that 30% of the neurons from the models' first two convolution layers are selected to be encrypted. Examples of possible combinations for exhaustive searches are calculated as follows:

$$\begin{aligned} C_{\text{LeNet-5(MNIST)}} &= \binom{6}{2} \times \binom{16}{5} = 6.522 \times 10^4 \\ C_{\text{VGG16}} &= \binom{64}{20} \times \binom{64}{20} = 3.849 \times 10^{32} \end{aligned} \tag{2}$$

We run exhaustive searches for all possible core neuron selections for the following scenarios: (1) when *up to* 2 neurons in the first layer and up to 6 neurons in the second layer of the LeNet-5 model for MNIST are selected to be encrypted; (2) when up to 2 neurons in the first layer and up to 4 neurons in the second layer of the modified LeNet-5 model for EMNIST are selected. For each $\{n_{11}, n_{21}\}$ pair, we present the number of possible selections in Table 2. For each core neuron selection, we evaluate the classification accuracy of authorized and unauthorized DNN evaluation (A_s and A_r). The best-case and the worst-case performances (in A_s, A_r, and s) are also reported in Table 2. For example, when 2 and 6 neurons from the first two convolution layers are selected for MNIST, there exist 120,120 possible selections. The best selection achieves a 99.26% accuracy for *authorized* DNN evaluation, i.e., a 0.1% performance drop. Meanwhile, it achieves a 77.32% accuracy for *unauthorized* DNN evaluation, i.e., a 22.04% performance drop.

Table 2: Ground truth of LeNet-5.

	$n_{1,1}$, $n_{2,1}$	Count	Best			Worst		
			s	A_s	A_r	s	A_s	A_r
MNIST [18]	1, 1	96	0.16	99.30	95.66	-0.63	93.36	96.14
	1, 2	720	0.37	99.30	89.30	-0.65	92.12	95.16
	1, 3	3360	0.41	99.28	86.04	-0.65	91.32	94.40
	1, 4	10920	0.44	99.30	83.98	-0.64	92.16	95.02
	1, 5	26208	0.47	99.32	80.86	-0.60	92.12	94.18
	1, 6	48048	0.48	99.34	76.64	-0.56	91.74	92.86
	2, 2	1800	0.24	98.66	87.32	-0.69	89.74	93.80
	2, 3	8400	0.31	98.86	85.36	-0.70	89.50	93.76
	2, 4	27300	0.39	99.06	82.24	-0.66	89.84	93.08
	2, 5	65520	0.42	99.18	82.32	-0.60	90.44	92.52
	2, 6	120120	0.46	99.26	77.32	-0.56	92.12	93.42
EMNIST [14]	1, 1	200	0.17	93.02	89.12	-0.52	83.26	83.75
	1, 2	1900	0.40	93.02	81.37	-0.48	83.85	83.38
	1, 3	11400	0.47	93.03	71.62	-0.43	84.39	83.01
	1, 4	48450	0.49	93.08	68.24	-0.36	84.55	81.78
	2, 2	8550	0.36	92.86	80.91	-0.70	60.01	64.13
	2, 3	51300	0.43	92.85	70.85	-0.65	59.60	62.76
	2, 4	218025	0.46	92.94	69.22	-0.55	68.87	69.92

Notes: **1.** A_o of MNIST and EMNIST are 99.36% and 93.08%, respectively. **2.** Count: the number of possible combinations with given $\{n_{11}, n_{21}\}$ pair.

Table 3: Random and greedy selections for LeNet-5.

	$n_{1,1}$, $n_{2,1}$	Count	Greedy		Random	
			#	T	Avg(BestRank)	T_g
MNIST [18]	1, 1	96	1	22	8.12	186.21
	1, 2	720	1	37	38.45	1495.43
	1, 3	3360	1	51	123.53	6235.05
	1, 4	10920	3	63	323.82	7260.88
	1, 5	26208	2	76	599.03	26699.64
	1, 6	48048	1	86	1052.65	101480.30
	2, 2	1800	1	39	79.19	3503.58
	2, 3	8400	1	53	292.28	17008.22
	2, 4	27300	1	66	750.19	58058.85
	2, 5	65520	1	78	1508.16	131966.71
	2, 6	120120	7	90	2488.02	34123.24
EMNIST [14]	1, 1	200	1	56	7.38	400.06
	1, 2	1900	1	94	41.40	3801.20
	1, 3	11400	1	133	165.04	24079.23
	1, 4	48450	3	169	566.97	31171.86
	2, 2	8550	1	105	153.07	16653.40
	2, 3	51300	1	142	683.22	99558.56
	2, 4	218025	6	176	2519.70	69996.76

For each $\{n_{11}, n_{21}\}$ pair in the ground truth dataset, we rank all the core neuron set selections by their quality score s and use it as a reference for future experiments. We have plotted the distribution of all 120,120 $\{A_s, A_r\}$ pairs for ($n_{1,1} = 2, n_{2,1} = 6$) for the MNIST dataset and LeNet-5 model, as shown in Figure 4. Since we assign a higher weight to A_s in the calculation of s, the top-ranked selections are mostly located on the right side of the plot, i.e., with higher A_s. A good selection algorithm is expected to find a highly ranked selection with significantly less computation than an exhaustive search. Note that we do not have the ground truth data for complex models, i.e., AlexNet, VGG16, and ResNet18, due to the excessive computation required for exhaustive searches.

Random and Greedy Selection. For the datasets with ground truth selection performance, we run the random selection (Algorithm 1) and the greedy selection (Algorithm 2) algorithms and report their performance in Table 3. For the greedy approach, we report the rank of the selected core neuron set as referenced to the ground truth results. We also report the time (in seconds) to complete the selection. As shown in the table, the greedy approach identifies the best core neuron sets in 13 out of 18 $\{n_{11}, n_{21}\}$ pairs, while the selections are very close to the best in the other five cases, e.g., ranked 3rd out of 10,920 selections (top 0.027%).

For each $\{n_{11}, n_{21}\}$ pair, we run the random selection algorithm for the same length of time that was used by the greedy approach, e.g., 22 seconds for $\{1, 1\}$ in MNIST. We pick the highest rank among all the random selections. We repeat this experiment 1000 times and report the average highest rank in Table 3. Moreover, we also keep the random selection experiments running and evaluate how much time it takes for random selection to identify a core set that ranks as high as the greedy approach. The result is reported as T_g in the table. As shown in the table, the greedy algorithm significantly outperforms the random selection approach.

Pruning-based Selection. We run the pruning-based selection algorithms for the datasets with ground truth selection performance and show their performance in Table 4. For each $\{n_{11}, n_{21}\}$ pair, we run four pruning algorithms and report the quality score s of each selected core neuron set, the rank of each selection (based on the ground truth in Table 2), the efficiency (T, in seconds) of the selection process, and the A_s and A_r of the protected DNN. For instance, when we designate one neuron from each of the first two convolution layers to be FHE-encrypted and use PFEC pruning to select these neurons, the network achieves 98.50% A_s (moderate performance) and 98.46% A_r (bad protection). The corresponding s score of -0.20 ranked 48 out of all 96 possible selections.

As shown in the table, the two static pruning methods, PFEC and FPGM, are extremely fast, while the dynamic pruning methods, HRank and GFS, are slow. The core neuron sets selected by all four algorithms demonstrate solid A_s in most cases, while A_r appears too high to provide enough protection. Since the dynamic pruning algorithms are significantly slower while they do not provide better selections, we do not consider them in the rest of the paper.

Pruning+Greedy Selection. Finally, we evaluate the performance of the Pruning+Greedy approach (Algorithm 4) using PFEC [52] and FPGM [34] pruning methods and compare the performance with the greedy approach. The performance on the smaller MNIST and EMNIST datasets is shown in Table 5. From the results, we have the following observations:

• **1.** The accuracy of authorized DNN evaluation, A_s, remains high (very close to A_o) for almost all $\{n_{11}, n_{21}\}$ pairs. The usability goal defined in Section 4.3 is always satisfied.

• **2.** The accuracy of unauthorized DNN evaluation, A_r, could be high when only a small number of neurons are encrypted. With the increase of n_{11} and n_{21} (i.e., more neurons in the first two convolution layers are protected), A_r decreases accordingly. In general, the unauthorized user only gets approximately 70% accuracy when five

Table 4: Performance of pruning-based selection algorithms for LeNet-5.

	$n_{1,1}$,	PFEC [52]					FPGM [34]					HRank [55]					GFS [80]				
	$n_{2,1}$	#	s	T	A_s	A_r	#	s	T	A_s	A_r	#	s	T	A_s	A_r	#	s	T	A_s	A_r
MNIST [18]	1,1	48	-0.20	3	98.50	98.46	44	-0.19	2	98.48	98.18	16	-0.03	384	99.24	99.16	50	-0.20	76	98.34	97.70
	1,2	229	-0.06	3	99.00	98.32	237	-0.06	2	98.88	97.72	172	-0.01	384	99.26	99.00	526	-0.21	128	98.28	97.48
	1,3	1531	-0.03	3	99.04	98.02	1273	0.00	2	99.02	97.36	1221	0.00	385	99.28	98.82	2670	-0.17	176	98.22	96.40
	1,4	6541	0.00	3	99.10	97.74	1936	0.16	2	99.12	94.40	6066	0.02	386	99.28	98.54	8239	-0.06	220	98.54	95.80
	1,5	15858	0.08	3	99.08	96.04	3301	0.28	2	99.26	92.22	14714	0.09	382	99.26	96.86	23347	-0.06	260	98.56	96.00
	1,6	27933	0.17	3	99.18	94.54	3146	0.37	2	99.24	88.14	29319	0.16	387	99.26	95.36	28929	0.16	295	98.50	89.14
	2,2	461	-0.13	3	98.04	94.64	313	-0.08	2	98.70	97.10	1127	-0.26	383	98.04	97.44	1247	-0.29	147	95.74	91.98
	2,3	2189	-0.03	3	98.16	93.02	1607	-0.01	2	98.90	96.76	5854	-0.19	384	98.26	97.00	5224	-0.16	191	95.68	88.12
	2,4	8576	0.02	3	98.26	92.34	1937	0.14	2	98.90	93.36	17152	-0.07	384	98.52	95.80	21766	-0.14	236	95.60	87.40
	2,5	11840	0.15	3	98.44	89.14	3934	0.23	2	98.96	90.92	16661	0.11	383	98.74	92.84	58709	-0.12	278	95.84	87.02
	2,6	27822	0.20	3	98.46	86.54	3966	0.33	2	98.94	86.10	20026	0.24	384	99.04	91.62	92539	0.01	312	95.82	75.02
EMNIST [14]	1,1	121	-0.28	3	91.71	91.35	64	-0.15	2	92.26	91.46	19	0.03	1105	92.78	90.75	25	0.20	213	93.02	92.35
	1,2	1071	-0.13	3	91.98	89.59	938	-0.11	2	92.27	90.51	434	0.03	1111	92.76	90.48	253	0.10	347	93.00	90.58
	1,3	4160	0.04	3	92.17	86.49	4963	0.01	2	92.36	88.62	3290	0.09	1101	92.80	89.50	597	0.28	479	92.97	85.64
	1,4	5885	0.30	3	92.35	74.88	8458	0.25	2	92.43	80.90	17987	0.13	1110	92.83	88.86	3919	0.34	601	93.02	84.32
	2,2	5408	-0.21	3	91.46	88.88	6540	-0.26	2	91.27	89.35	2179	-0.08	1108	92.22	89.62	5621	-0.22	408	85.30	78.94
	2,3	7320	0.01	3	91.65	84.28	41379	-0.12	2	91.48	86.93	24419	-0.06	1130	92.16	88.85	16559	-0.03	537	86.78	68.88
	2,4	4823	0.27	3	92.19	73.17	16401	0.16	2	91.80	77.82	118606	-0.02	1118	92.17	88.04	122706	-0.02	656	86.38	67.17

neurons from the first two convolution layers (30 neurons in total) of the LeNet-5 model for EMNIST are encrypted. The security goal defined in Section 4.3 is also satisfied.

- **3.** The Pruning+Greedy mechanism saves core neuron selection time (T) significantly when $n_{i1} << n_i$, i.e., when a relatively small number of neurons in a layer will be encrypted. However, the improved efficiency comes at the cost of selection performance. Meanwhile, there does not exist any obvious differences in the performance of PFEC and FPGM.

Finally, we report the performance of greedy and pruning+greedy approaches on the three complex datasets, GTSRB, CIFAR10, and Tiny-ImageNet. We evaluate these algorithms when 15% to 75% of the neurons in the first two convolution layers are selected to be encrypted. Since we do not have the ground truth (ranking) for all possible core neuron sets, we only report A_s, A_r, and T in Table 6. As shown, even if we select 25% neurons from the first two convolutional layers, all the models' usability and security goals are clearly satisfied. Since we use $p = 2$ for pruning+greedy, when 50% or more of the neurons are to be encrypted, there is no need for pruning. The pruning+greedy approach provides slightly worse selections while being more efficient. In summary, the pruning+greedy approach balances selection efficiency and performance compared to the greedy algorithm. However, since neuron selection is a one-time process, the model owner may want to employ the greedy approach to obtain better performance while accepting the overhead.

5.3 Computation for DNN Evaluation

The essential advantage of employing partial DNN encryption in PrivDNN is to reduce the heavy-lifting ciphertext operations. Instead of encrypting the full network and performing the entire DNN evaluation process using FHE in ciphertext, PrivDNN only protects the *core* of the target DNN, i.e., the neurons that are essential to the performance. Only a subset of the neurons are encrypted, and the corresponding operations are performed in the ciphertext domain, while the majority of the neurons and the corresponding operations are kept intact in the plaintext domain.

Table 5: Performance of the Pruning+Greedy approach for MNIST and EMNIST.

$n_{1,1}$,	Greedy				Pruning+Greedy							
					PFEC [52]				FPGM [34]			
$n_{2,1}$	#	T	A_s	A_r	#	T	A_s	A_r	#	T	A_s	A_r
					MNIST [18]							
1,1	1	22	99.30	95.66	9	5	99.26	98.84	5	6	99.28	98.68
1,2	1	37	99.30	89.30	84	10	99.32	98.54	48	11	99.34	97.92
1,3	1	51	99.28	86.04	437	18	99.36	97.34	322	19	99.34	96.46
1,4	3	63	99.22	81.12	1430	29	99.32	94.98	1430	30	99.32	94.98
1,5	2	76	99.26	77.66	824	43	99.24	88.88	824	44	99.24	88.88
1,6	1	86	99.34	76.64	2296	60	99.22	87.02	2296	61	99.22	87.02
2,2	1	39	98.66	87.32	282	12	98.50	95.76	262	14	98.62	96.28
2,3	1	53	98.86	85.36	754	20	98.60	94.00	417	22	98.62	93.18
2,4	1	66	99.06	82.24	906	31	98.92	92.04	833	32	98.78	90.62
2,5	1	78	99.18	82.32	147	46	99.12	87.44	147	46	99.12	87.44
2,6	7	90	99.22	80.82	59	63	99.12	80.80	59	63	99.12	80.80
					EMNIST [14]							
1,1	1	56	93.02	89.21	16	9	93.04	92.04	20	11	93.02	92.21
1,2	1	94	93.02	81.37	3	20	93.00	83.57	15	21	93.03	86.23
1,3	1	133	93.03	71.62	7	37	93.04	77.58	3	38	93.03	73.92
1,4	3	169	93.06	66.23	64	59	93.04	73.64	5	60	93.05	67.06
2,2	1	105	92.86	80.91	6	25	92.79	83.35	11	26	92.91	85.64
2,3	1	142	92.85	70.85	36	42	92.76	77.12	2	43	92.88	73.33
2,4	6	176	92.88	63.81	343	65	92.77	72.99	6	65	92.88	63.81

We define the DNN encryption ratio $ER = \frac{N_e}{N}$, i.e., the proportion of core neurons out of all neurons in the DNN. We also define the encryption ratio of addition and multiplication operations as ER_+ and ER_*, denoting the proportion of ciphertext addition and multiplication operations out of all operations, respectively. Compared to FHE operations, the computation costs of plaintext operations

Table 6: Performance of the Pruning+Greedy approach for GTSRB, CIFAR10, and Tiny-ImageNet.

$n_{i,1}$ (%)	Greedy		Pruning+Greedy			
			PFEC [52]		FPGM [34]	
	T	A_s A_r	T	A_s A_r	T	A_s A_r
GTSRB [38]						
15	35703	94.05 89.33	8656	93.90 86.94	8798	93.94 86.33
25	56082	94.17 79.90	24665	94.06 79.16	24560	94.00 77.83
50	104842	94.27 68.95	-	-	-	-
75	121636	94.25 37.96	-	-	-	-
CIFAR10 [47]						
15	2924	90.90 64.20	679	89.82 53.36	686	90.14 53.20
25	4899	91.18 60.34	2121	90.32 66.72	2144	90.32 66.72
50	8336	90.78 55.90	-	-	-	-
75	10479	90.52 24.90	-	-	-	-
Tiny-ImageNet [59]						
15	3954	72.10 44.84	911	71.10 39.06	920	70.96 37.94
25	6626	72.16 43.18	2842	71.48 39.30	2889	71.18 42.52
50	11388	72.02 45.66	-	-	-	-
75	14289	72.04 26.76	-	-	-	-

Notes: **1.** A_o of GTSRB and CIFAR10 are 93.24% and 90.22%, respectively.

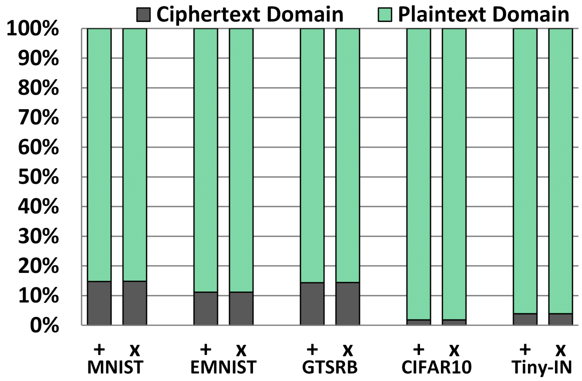


Figure 5: Distribution of addition (+) and multiplication (×) operations in the ciphertext and plaintext domains.

are negligible. Meanwhile, the multiplication operations (usually followed by a rescale operation), cost more than ten times time than the addition operations. Therefore, the inference time of PrivDNN is almost linear to ER_*, which is mostly determined by ER.

In Figure 5, we demonstrate the distribution of operations (addition and multiplication) in the ciphertext and plaintext domains. For MNIST and EMNIST, we use $\{n_{1,1} = 1, n_{2,1} = 3\}$. For the larger models, we encrypt 50% of the neurons in the first two convolution layers. As shown in Figure 5, PrivDNN only places a very small portion of the operations in the ciphertext domain, while the majority of the operations remain in plaintext. Therefore, compared with the full DNN encryption approaches in the literature, e.g., [26], PrivDNN saves 85% to 98% of the cipher computation. Besides, the traditional full DNN encryption approach requires a higher HE noise budget, slowing down all cipher operations. For example, when we implement LeNet-5 on the MNIST dataset, the PrivDNN inference for a batch of 5,000 testing images costs 190 seconds with the selection of $\{n_{1,1} = 1, n_{2,1} = 4\}$. In comparison, the full DNN encryption approach takes 5,647 seconds. That is, PrivDNN achieves a 29.7x speedup with $ER_* = 18.75\%$.

Besides the efficiency benefits, PrivDNN also saves a significant amount of memory so that the deployment of FHE-based inference for large-scale DNN models becomes practical for commodity desktop computers. The traditional full DNN encryption approach needs a higher noise budget to support ciphertext operations along all the DNN layers, which causes the cipher to take up more memory for each ciphertext value. For example, the ciphertext weights of the first fully connected layer of LeNet-5/MNIST take 151 GB, which is beyond the capacity of most PCs. On the contrary, PrivDNN uses approximately 2.5 GB of memory with $\{n_{1,1} = 1, n_{2,1} = 3\}$, and 5 GB with $\{n_{1,1} = 2, n_{2,1} = 6\}$. Therefore, PrivDNN dramatically reduces the memory utilization for FHE-based DNN inference and consequently enables the deployment of secure DNN inference for resource-constrained users such as small businesses.

Summary of Experimental Results. From the experiments, we conclude that: (1) The greedy algorithm effectively identifies a core neuron set that achieves high accuracy for authorized DNN execution and low accuracy for unauthorized execution. (2) Encrypting approximately 15% to 25% of the neurons in the first two convolution layers of a large DNN effectively reduces its accuracy by 20% for unauthorized users, i.e., the DNN becomes practically unusable for them. (3) With the partial DNN encryption scheme, only a few DNN evaluation operations are conducted in ciphertext. Compared with the full-DNN encryption scheme, PrivDNN saves the cipher operations by 85% to 98%, reduces the inference time and memory usage by 97% on smaller models and much more on larger models.

6 Security Analysis

We discuss the security/privacy guarantees of the proposed PrivDNN framework. We consider two aspects of privacy: the model owners would like to protect their proprietary deep-learning models, while the data owners would like to protect their testing samples.

6.1 Model Privacy

Privacy Expectations. The downstream user, i.e., the data owner C, should not be able to utilize the partially encrypted DNN to reconstruct a model to achieve comparable accuracy to the original model. We discuss the following aspects: (1) information that is disclosed to the data owner C, (2) the capability of honest data owners, (3) the capability of curious data owners, and (4) the capability of dishonest data owners and the potential defense.

Structure and Parameters. PrivDNN allows C to run some inference operations in the plaintext domain to avoid expensive cipher domain operations. Therefore, $\mathcal{S}$ must share the model structure and the non-core parameters in plaintext with C. $\mathcal{S}$ also shared the core parameters in ciphertext with C.

The Honest Data Owners. C receives the core neuron weights in ciphertext. As explained in Section 3.2, the confidentiality of the encrypted weights is guaranteed by the security of the FHE algorithm CKKS. It is proved that the security CKKS relies on the hardness of Ring-LWE, which indicates that if there is an adversary who can break CKKS, it can be used to tackle the RLWE problem. It is believed that there is only a negligible probability of this. In this regard, C cannot break the encryption to learn the weights, and hence, it could only achieve A_r with the non-core parameters. The core neuron selection algorithms ensure that A_r is always much lower than A_s (accuracy for the authorized users). This is also demonstrated by experiments in Tables 5 and 6.

The Curious Data Owners and the Model Recovery Attack. A curious data owner (C) precisely follows the SMC protocol in model execution, but she still attempts to infer knowledge about the protected model from the information she received and the intermediate results. We define the model recovery attack as follows: the attacker (curious data owner) knows the target model's architecture and a subset of plaintext weights, while the remaining weights are encrypted. According to the threat model (Section 3.2), the attacker cannot break the encryption, nor does she have the model owner's proprietary data to train the model. The attacker uses a small number of labeled samples to *retrain* the model in an attempt to recover the unknown weights and restore the model. In particular, to get the model with comparable accuracy to the original model, the attacker executes the model recovery attack as follows: (1) The attacker removes all the encrypted weights from the protected DNN and reset them. (2) With her local samples, the attacker tunes the entire model or freezes the plaintext parameters and only trains the missing weights. Our experiments prove that the first scenario is always better, so we adapt the first scenario.

To evaluate the attacker's ability, we run the experiments with two settings: [Setting I] Only a smaller amount of neurons are encrypted to provide less protection with high efficiency: $n_{1,1} = 1, n_{2,1} = 2$ for MNIST and EMNIST, 50% of the neurons in the first two convolution layers for GTSRB and CIFAR10. [Setting II] More neurons are encrypted for better protection: $n_{1,1} = 2, n_{2,1} = 4$ for MNIST and EMNIST, 75% of the neurons in the first two convolution layers for GTSRB and CIFAR10. (still only 2.27% of all convolution neurons for CIFAR10). To tune the model, we use the same parameters as the original training process except for a one-tenth learning rate for the selected neurons and 64 epochs.

The experimental results for the recovery attacks are presented in Table 7, where A_{rec} denotes the accuracy of the recovered models. Results show that DNNs protected by PrivDNN are relatively resilient against recovery attacks. In particular: (1) In most cases, the models recovered from partially encrypted DNNs perform noticeably worse than the protected models. Note that, model accuracy does not linearly increase with training resources/efforts. Training a moderately accurate model from a modest amount of data is relatively easy, while it is exponentially more challenging to improve the accuracy for another 5% or even 3% from a (publicly available) modest model. Therefore, a small accuracy drop of the recovered models dramatically reduces their values. (2) In some cases, the recovered models perform worse than models trained from scratch with 1,000 samples (A_t). As discussed in the threat model, the data owners are not expected to have too many samples for the model recovery attack. Otherwise, they could train the model by themselves. (3) Large models are more valuable to protect in real-world practice. For large models such as VGG16 for CIFAR10, PrivDNN could encrypt a relatively larger portion of the first two convolution layers, making it highly challenging for the data owners to recover the network. Meanwhile, since the networks are deep and complex, most of the computations are still in plaintext, even when the first two convolution layers are mostly encrypted.

Table 7: Recover attack by curious data owners.

Model recovery from partially encrypted DNN												
	Setting 1						Setting 2					
	A_s	A_r	A_{rec}				A_s	A_r	A_{rec}			
			100	250	500	1000			100	250	500	1000
M	99.3	89.3	89.1	89.2	89.5	90.5	99.1	82.2	92.8	93.0	93.3	94.4
E	93.0	81.4	81.5	84.9	88.7	89.8	92.9	63.8	66.1	74.9	83.9	86.1
G	94.3	69.0	86.1	89.7	91.8	92.7	94.3	38.0	68.7	79.7	86.1	90.4
C	90.8	55.9	73.4	78.9	80.6	83.6	90.5	24.9	21.8	19.7	48.2	58.2

A_t									
	100	250	500	1000		100	250	500	1000
M	59.8	78.3	85.9	91.8	E	29.4	44.7	58.6	68.5
G	27.6	49.9	72.5	91.5	C	23.5	31.2	39.5	49.9

Notes: **1.** 100, 250, 500, and 1000: the number of pictures to train the model to recover encrypted weights. **2.** A_{rec}: accuracy of the recovered model. 3. A_t: accuracy of training from scratch.

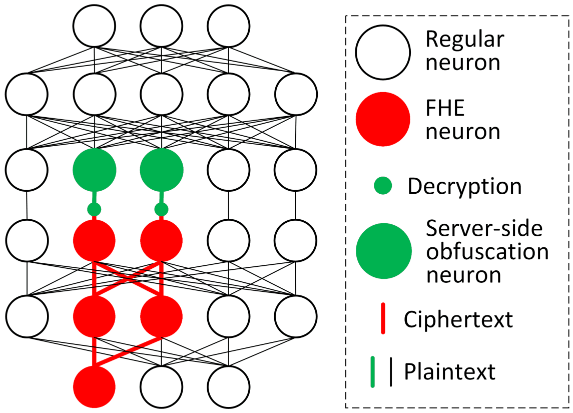


Figure 6: PrivDNN with a server-side obfuscation layer.

6.2 Data Privacy

Privacy Expectations. The model owner, $\mathcal{S}$, should not be able to reconstruct the raw pixels of any testing image or to recover the visual features of any testing image. We discuss the following aspects: (1) information that is (not) disclosed to the model owner, (2) the capability of the honest-but-curious model owners, and (3) the capability of dishonest model owners and the potential defense.

Raw Testing Samples. In the design of PrivDNN, most inference operations are executed on the data owner (C) side. Therefore, the raw pixel space representations of the testing samples never leave C, and they are considered secure.

The Curious Model Owners and the Sample Inference Attack. When C sends the encrypted output of the core neurons to $\mathcal{S}$ for decryption, $\mathcal{S}$ learns the decrypted values of the intermediate DNN outputs. A curious $\mathcal{S}$ will attempt to utilize the intermediate outputs to infer features of the data samples. This is highly challenging, if not impossible, since (1) she only sees a subset of outputs from the last partially encrypted layer, and (2) C will follow the protocol to obfuscate the intermediate results before sending them for decryption. In particular, we have adopted two DNN input sample recovery attacks in the literature, split learning [23] and autoencoder [11], in an attempt to recover C's samples from the obfuscated plaintext output of the core neurons. As shown in Figure 8 and 9, the attacks could not succeed even when 100% of the Layer 2 output (obfuscated) is known to $\mathcal{S}$.

6.3 The Dishonest Participants

The Threat Model Revisited. We would like to note that the honest-but-curious model is adopted in this explorative project on SMC for deep learning. However, the honest assumption could be too strong for real-world adoption. Here, we attempt to relax this assumption to explore the simple attacks from malicious parties and propose corresponding controls. We acknowledge that more research efforts need to be devoted to further investigating the complex attacks and defenses to fully relax the honest assumption.

The Dishonest Data Owners and the Weight-Stealing Attack. A malicious C may break the honest-but-curious assumption to launch a *weight-stealing attack*: it sends the encrypted and obfuscated model parameters to $\mathcal{S}$, who decrypts and returns the weighs. Hence, C will get the whole plaintext model.

To defend against the attack, we infuse a server-side obfuscation layer after the last partially encrypted layer, as shown in Figure 6. The obfuscation layer is a convolutional layer (5×5 kernels) with only one input and output channel. The weights for the neurons directly succeeding the encrypted core neurons are kept with $\mathcal{S}$, while the weights for the other neurons in this layer are shared in plaintext with C. This layer is used to obfuscate the decryption results before returning them to C, that is, $\mathcal{S}$ decrypts the intermediate results, performs convolution operations with the server-side obfuscation neurons, and returns the obfuscated results to C. Since C does not have the parameters of the obfuscation neurons, even if she sends the encrypted model weights to $\mathcal{S}$, C cannot recover such weights after server-side obfuscation. More importantly, the obfuscation layer is polymorphic, i.e., when we freeze the entire network, reset and retrain the obfuscation neurons (the green neurons in Figure 6), we obtain a different set of the obfuscation parameters in each retrain. The polymorphism feature makes it impossible for a malicious C to reverse engineer the parameters of the obfuscation neurons. Note that the computation of the obfuscation layer is negligible since it is only a convolution layer in plaintext. In our experiments, this polymorphic obfuscation layer only decreases the model's accuracy by no more than 0.1%. Server-side obfuscation is also compatible with client-side obfuscation. In client-side obfuscation, C multiplies all the encrypted values from the same channel by a random value σ before sending them to $\mathcal{S}$. Since the operations in the server-side obfuscation (convolution) layer are all linear, C could de-obfuscate the results by multiplying by $1/\sigma$.

Dishonest Model Owner. A malicious $\mathcal{S}$ may use engineered core weights (e.g., using all 1s and 0s) to obtain (partial) plaintext input that was sent into the core neurons. This is prevented by data obfuscation. Besides, with the malicious core weights, A_s will dramatically decrease, and C easily detects the malicious model.

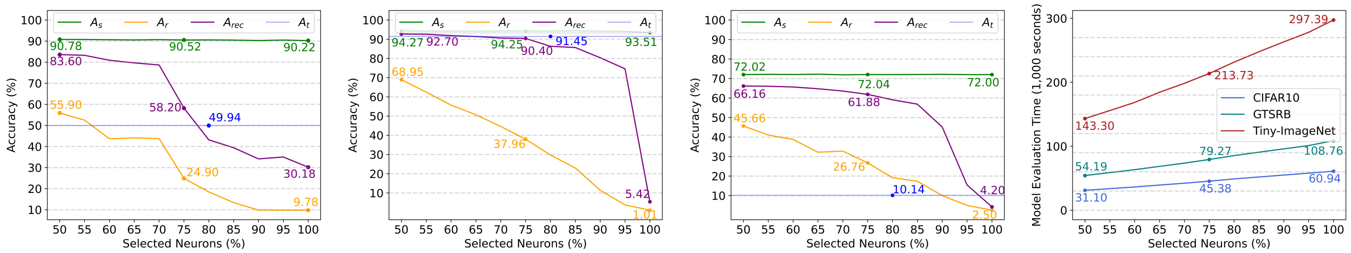
6.4 The Performance-Security Trade-off

In PrivDNN, $\mathcal{S}$ has full control of the selection of the core neurons. In practice, $\mathcal{S}$ identifies $n_{i,1}$, which is the number of core neurons to be selected from layer i. PrivDNN then selects the (near) optimal set of core neurons that achieve the best performance for authorized users (A_s) and the least performance for unauthorized users (A_r). On the other hand, the choice of parameters $n_{i,1}$ implies a trade-off between performance and security. If $\mathcal{S}$ chooses to encrypt fewer core neurons (smaller $n_{i,1}$), it saves computation at C but provides weaker protection, i.e., a higher A_r and potentially more vulnerable against the model recovery attack (higher A_{rec}). On the contrary, if $\mathcal{S}$ encrypts more core neurons, it will require higher computation at C but provide better protection for $\mathcal{S}$'s model.

To demonstrate the performance-security trade-off, we evaluate PrivDNN on three datasets: CIFAR10+VGG16, GTSRB+AlexNet, and Tiny-ImageNet+ResNet18. We protect 50% to 100% of the neurons from the first two convolution layers. The results are shown in Figure 7. We can observe the following: (1) we achieve consistently high A_s (accuracy for authorized model evaluation), which is always very close to the original model accuracy (A_o). (2) A_r (accuracy for unauthorized evaluation) decreases with the increased number of encrypted neurons, i.e., from 55.90% to 9.78% for CIFAR10, from 68.95% to 1.01% for GTSRB, and from 45.66% to 2.50% for Tiny-ImageNet. (3) Completely retraining the model with 1000 samples achieves subpar accuracy (A_t): 49.94% for CIFAR, 91.45% for GTSRB, and 10.14% for Tiny-ImageNet. (4) The effectiveness of the model recovery attack also decreases, especially when ~100% of the first two layer neurons are protected. When 50%, 75%, and 100% of the first two layers are encrypted, the accuracy of the model recovered with 1,000 samples (A_{rec}) decreases from 83.60%, 58.20% to 30.18% for CIFAR-10, from 92.70%, 90.40%, to 5.42% for GTSRB, from 66.16%, 61.88% to 4.20% for Tiny-ImageNet. (5) A_{rec} almost always stays in between A_s and A_r, i.e., the model recovery attack could improve the performance of the broken models by tuning them with 1,000 samples. However, they cannot reach the accuracy of the original model, and they fail badly when a relatively larger proportion of the first two layer neurons are protected. (6) The model recovery attack appears more effective on GTSRB. This is explained by the fact that GTSRB has highly similar images in each class, so a classifier could easily achieve high accuracy with only a small number of training samples. As shown in Figure 7 (B), A_t is very high (approximately 3% below A_o), and A_{rec} stays lower than A_t, i.e., with the same number of labeled samples, it is easier to train a model from scratch than to recover a protected model.

Last, the DNN inference time increases linearly with the number of core neurons, which is consistent with the discussions in Section 5.3. Note that invoking an FHE-based full-model encryption scheme for a large model like VGG16 is extremely expensive. Hence, the overhead for PrivDNN is still considered very acceptable.

To balance the efficiency (for C) and model security (for $\mathcal{S}$), the model owner could invoke PrivDNN to evaluate the model with different $n_{i,1}$ settings and empirically pick the one that satisfies her privacy goals while requires reasonable computation for C. Finally, we acknowledge that more effective model recovery attacks may emerge in the future as part of the cat-and-mouse nature of attacks and defenses in cybersecurity. A possible mitigation is to further increase the proportion of the protected neurons. For large models, such as VGG16 and ResNet18, the first two convolution layers contain less than 5% of all model weights. Encrypting all of them would significantly reduce A_{rec} while still maintaining a low DNN encryption ratio (ER), i.e., maintaining high efficiency.



(A) CIFAR10+VGG16 **(B) GTSRB+AlexNet** **(C) Tiny-ImageNet+ResNet18** **(D) Model evaluation time**

Figure 7: The performance-security trade-off: the distribution of A_s, A_r when 50% to 100% neurons are encrypted. The corresponding A_{rec} and A_t when 1,000 samples are used for model recovery attacks or train the model for (A) CIFAR10+VGG16, (B) GTSRB+AlexNet, and (C) Tiny-ImageNet+ResNet18. (D): Model evaluation time (T) for a batch of up to 8192 testing samples.

7 Related Works

Privacy-preserving DNN Evaluation with Data Encryption. Machine learning applications usually involve two steps: training the model from labeled data and inference for unlabeled samples using the trained model. Most of the existing secure ML works focus on protecting the data, i.e., the data owner wants to evaluate the testing samples while not leaking the data to the untrusted model owner. Hence, the data owner encrypts the testing samples using FHE before sending them to the model owner.

CryptoNets [26] is the first work to apply FHE to secure DNN evaluation. It takes 250 seconds to evaluate a batch of encrypted MNIST images. Later works attempted to improve the efficiency of secure DNN evaluation and support more complex models with higher accuracy. LoLa [7] encrypts entire layers and changes data representations throughout the computation to get an 11.2× speedup on CIFAR10. Faster CryptoNets [13] improved performance over CryptoNets by leveraging sparsity properties. CHET [16] provided an optimizing compiler for FHE DNN inference to offer a high-level user framework to automate parameter tuning. Recently, more efforts have been devoted to improving the activation functions. CryptoDL [36] explored more common activations besides the square function. FHE–DiNN [5] use a precomputed table to process the cipher activation whose complexity is strictly linear in the depth of the network and whose parameters can be set beforehand. However, it can only be applied to integer ciphers, which means its accuracy is still limited. More works try to use low-degree polynomials to approximate the non-linear activation functions. [9] combine CryptoNets and batch normalization principle. It normalizes the convolutional output before activation and uses different degrees of polynomials to imitate ReLU. [61] examines the contribution of different properties, such as differentiable, continuous, monotonic, etc., to guide the selection of polynomial functions. [51] selects optimal composite polynomials for the sign function and uses the sign function to implement ReLU.

Privacy-preserving DNN Evaluation with Model Encryption. Compared with data protection, fewer works focus on model protection. [4] first studied the protection of the private DNN models, while [43] focused on the protection of CNNs. They adopt the same data protection schemes for DNN model protection, i.e., they encrypt all the parameters in the model and feed the encrypted model with plaintext or ciphertext data to perform DNN evaluation entirely in ciphertext. Unlike these approaches, PrivDNN is the first to observe and utilize a key feature in DNN model protection that is unnecessary to protect all the parameters in a large DNN. Therefore, PrivDNN is able to achieve a significant boost in the efficiency of privacy-preserving DNN evaluation by only encrypting a subset of essential neurons of the entire DL model.

DNN Model Recovery Attacks. Model recovery (stealing or extraction) attacks aim to obtain model hyperparameters, architecture, or trained weights, [10, 62, 78]. The weight-stealing attacks are the most similar to our model recovery attacks. They can be grouped into two categories: (1) stealing exact properties and (2) stealing approximate behavior [62]. [57] proposes the first exact model weight stealing attack on a binary classifier. [68] extended the idea and presented an equation-solving attack for support vector regression. In outsourced DNN execution, weights could be stolen from the computing platforms or the communication channel [37, 39, 78]. Such attacks are not applicable in PrivDNN, since the cryptography primitives are assumed to be unbreakable to the attacker. Meanwhile, in query-based approximate model stealing, the attacker queries the target model with input images to learn the predictions and then trains a local model with learned image-prediction pairs [63]. Optimization techniques, such as active learning, reinforcement learning, and evolutionary algorithms, are employed to save query budgets [10, 65, 66]. However, those attacks still need significantly more labeled samples than our model recovery attack. For example, [71] stole a ResNet18 model (original accuracy is 78.52%) and achieved 72.83% clone accuracy with the entire CIFAR10 dataset (50K samples) as a proxy, or 43.56% with synthetic fake data. [68] stole a VGG16 model and achieved 93.7 - 98.6% clone accuracy of the original model accuracy with over 3M natural pictures. [44] stole a GTSRB[38] classifying model and achieved 97.9% clone accuracy of the original model accuracy with 102K synthetic samples.

8 Conclusion

We present PrivDNN, a practical framework to protect DNN models in privacy-preserving DNN evaluation. With the novel partial DNN encryption scheme, the authorized DNN evaluation accuracy remains very close to the original DNN accuracy, while the unauthorized users get significantly decreased accuracy. We design and implement three algorithms to identify the core neurons from DNN models for effective protection. As shown in extensive experiments on five popular benchmarking datasets and DNN models, PrivDNN reduces the inference time by up to 29.7x in privacy-preserving model evaluation while keeping the model safe. We share the code at: https://github.com/LiangqinRen/PrivDNN.

Acknowledgments

Liangqin Ren, Zeyan Liu, Fengjun Li, and Bo Luo were supported in part by NSF IIS-2014552, DGE-1565570, DGE-1922649, and the Ripple University Blockchain Research Initiative. Kaitai Liang was supported in part by the European Union's Horizon Europe Research and Innovation Programme under Grant No. 101073920 (TENSOR), No. 101070052 (TANGO), and No. 101070627 (REWIRE). The authors would like to thank the anonymous reviewers and the shepherd for their valuable comments and suggestions.

References

[1] Ahmad Al Badawi, Chao Jin, Jie Lin, Chan Fook Mun, Sim Jun Jie, Benjamin Hong Meng Tan, Xiao Nan, Khin Mi Mi Aung, and Vijay Ramaseshan Chandrasekhar. 2020. Towards the alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus. *IEEE Transactions on Emerging Topics in Computing* 9, 3 (2020), 1330–1343.

[2] Mohammad Al-Rubaie and J Morris Chang. 2019. Privacy-preserving machine learning: Threats and solutions. *IEEE Security & Privacy* 17, 2 (2019), 49–58.

[3] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjosteen, Angela Jäschke, Christian A Reuter, and Martin Strand. 2015. A guide to fully homomorphic encryption. *Cryptology ePrint Archive* (2015).

[4] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2014. Machine learning classification over encrypted data. *Cryptology ePrint Archive* (2014).

[5] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast homomorphic evaluation of deep discretized neural networks. In *38th Annual International Cryptology Conference (CRYPTO)*. Springer, 483–512.

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)* 6, 3 (2014), 1–36.

[7] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. 2019. Low latency privacy preserving inference. In *International Conference on Machine Learning*. PMLR, 812–821.

[8] Linhang Cai, Zhulin An, Chuanguang Yang, Yangchun Yan, and Yongjun Xu. 2022. Prior gradient mask guided pruning-aware fine-tuning. In *AAAI*, Vol. 36. 140–148.

[9] Hervé Chabanne, Amaury De Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. 2017. Privacy-preserving classification on deep neural network. *Cryptology ePrint Archive* (2017).

[10] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. 2020. Exploring connections between active learning and model extraction. In *29th USENIX Security Symposium (USENIX Security 20)*. 1309–1326.

[11] Shuangshuang Chen and Wei Guo. 2023. Auto-Encoders in Deep Learning—A Review with New Perspectives. *Mathematics* 11, 8 (2023), 1777.

[12] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT)*. Springer, 409–437.

[13] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. 2018. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953* (2018).

[14] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. 2017. EMNIST: Extending MNIST to handwritten letters. In *International joint conference on neural networks (IJCNN)*. IEEE, 2921–2926.

[15] Jack LH Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. 2018. Doing real work with FHE: the case of logistic regression. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 1–12.

[16] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. 2019. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *ACM PLDI*. 142–156.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[18] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[19] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems* 27 (2014).

[20] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.

[21] Wenliang Du and Mikhail J Atallah. 2001. Secure multi-party computation problems and their applications: a review and open problems. In *Proceedings of the 2001 workshop on New security paradigms*. 13–22.

[22] Abigail English and Carol A. Ford. 2004. The HIPAA Privacy Rule and Adolescents: Legal Questions and Clinical Challenges. *Perspectives on Sexual and Reproductive Health* 36, 2 (2004), 80–86. http://www.jstor.org/stable/3181198

[23] Ege Erdoğan, Alptekin Küpçü, and A Ercüment Çiçek. 2022. Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society*. 115–124.

[24] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[25] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[26] 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 *International conference on machine learning*. PMLR, 201–210.

[27] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 315–323.

[28] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.

[29] Rafael Rodrigo Guillén, Higinio Mora Mora, and Jorge Azorín-López. 2022. A Review of Deep Learning Methods for Detection of Gatherings and Abnormal Events for Public Security. In *International Conference on Ubiquitous Computing and Ambient Intelligence*. Springer, 809–814.

[30] Jun Han and Claudio Moraga. 1995. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *From Natural to Artificial Neural Computation: International Workshop on Artificial Neural Networks*. 195–201.

[31] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* 28 (2015).

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[33] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. 2018. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866* (2018).

[34] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*. 4340–4349.

[35] Yang He and Lingao Xiao. 2023. Structured Pruning for Deep Convolutional Neural Networks: A survey. *arXiv preprint arXiv:2303.00566* (2023).

[36] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189* (2017).

[37] Sanghyun Hong, Michael Davinroy, Yigitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitras. 2018. Security analysis of deep neural networks operating in the presence of cache side-channel attacks. CoRR abs/1810.03487 (2018). *arXiv preprint arXiv:1810.03487* (2018).

[38] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. 2013. Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*.

[39] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. 2020. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 385–399.

[40] Zehao Huang and Naiyan Wang. 2018. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*. 304–320.

[41] Alberto Ibarrondo and Melek Önen. 2018. Fhe-compatible batch normalization for privacy preserving deep learning. In *ESORICS Workshops on Data Privacy Management, Cryptocurrencies and Blockchain Technology*. 389–404.

[42] Takumi Ishiyama, Takuya Suzuki, and Hayato Yamana. 2020. Highly accurate CNN inference using approximate activation functions over homomorphic encryption. In *IEEE International Conference on Big Data (Big Data)*. 3989–3995.

[43] Nayna Jain, Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Uttam Kumar. 2021. Efficient CNN building blocks for encrypted data. *arXiv preprint arXiv:2102.00319* (2021).

[44] 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)*. IEEE, 512–527.

[45] Barry L Kalman and Stan C Kwasny. 1992. Why tanh: choosing a sigmoidal function. In *International Joint Conference on Neural Networks*, Vol. 4. IEEE, 578–581.

[46] Justin Ker, Lipo Wang, Jai Rao, and Tchoyoson Lim. 2017. Deep learning applications in medical image analysis. *Ieee Access* 6 (2017), 9375–9389.

[47] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

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

[50] Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems* 2 (1989).

[51] Joon-Woo Lee, Hyungchul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. 2022. Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access* 10 (2022), 30039–30054.

[52] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).

[53] Shutao Li, Weiwei Song, Leyuan Fang, Yushi Chen, Pedram Ghamisi, and Jon Atli Benediktsson. 2019. Deep learning for hyperspectral image classification: An overview. *IEEE Transactions on Geoscience and Remote Sensing* 57, 9 (2019), 6690–6709.

[54] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. 2017. Runtime neural pruning. *Advances in neural information processing systems* 30 (2017).

[55] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. 2020. Hrank: Filter pruning using high-rank feature map. In *CVPR*. 1529–1538.

[56] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).

[57] Daniel Lowd and Christopher Meek. 2005. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 641–647.

[58] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. 2021. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence* 44, 7 (2021), 3523–3542.

[59] Mohammed Ali mnmoustafa. 2017. Tiny ImageNet. https://kaggle.com/competitions/tiny-imagenet

[60] Jianjun Ni, Yinan Chen, Yan Chen, Jinxiu Zhu, Deena Ali, and Weidong Cao. 2020. A survey on theories and applications for self-driving cars based on deep learning methods. *Applied Sciences* 10, 8 (2020), 2749.

[61] Srinath Obla, Xinghan Gong, Asma Aloufi, Peizhao Hu, and Daniel Takabi. 2020. Effective Activation Functions for Homomorphic Evaluation of Deep Neural Networks. *IEEE Access* 8 (2020), 153098–153112.

[62] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. 2023. I know what you trained last summer: A survey on stealing machine learning models and defences. *Comput. Surveys* (2023).

[63] 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.

[64] Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer. 2020. Deep learning for financial applications: A survey. *Applied Soft Computing* 93 (2020), 106384.

[65] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. 2020. Activethief: Model extraction using active learning and unannotated public data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 865–872.

[66] Li Pengcheng, Jinfeng Yi, and Lijun Zhang. 2018. Query-efficient black-box attack by active learning. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1200–1205.

[67] Prajit Ramachandran, Barret Zoph, and Quoc V Le. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941* (2017).

[68] Robert Nikolai Reith, Thomas Schneider, and Oleksandr Tkachenko. 2019. Efficiently stealing your machine learning models. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*. 198–210.

[69] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. 1978. On data banks and privacy homomorphisms. *Foundations of secure computation* 4, 11 (1978), 169–180.

[70] Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems* 33 (2020), 20378–20389.

[71] Sunandini Sanyal, Sravanti Addepalli, and R Venkatesh Babu. 2022. Towards data-free model stealing in a hard label setting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 15284–15293.

[72] SEAL 2022. Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[73] Dinggang Shen, Guorong Wu, and Heung-Il Suk. 2017. Deep learning in medical image analysis. *Annual review of biomedical engineering* 19 (2017), 221–248.

[74] Li Shen, Laurie R Margolies, Joseph H Rothstein, Eugene Fluder, Russell McBride, and Weiva Sieh. 2019. Deep learning to improve breast cancer detection on screening mammography. *Scientific reports* 9, 1 (2019), 12495.

[75] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[76] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[77] Jinjiang Wang, Yulin Ma, Laibin Zhang, Robert X Gao, and Dazhong Wu. 2018. Deep learning for smart manufacturing: Methods and applications. *Journal of manufacturing systems* 48 (2018), 144–156.

[78] Mengjia Yan, Christopher W Fletcher, and Josep Torrellas. 2020. Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures. In *29th USENIX Security Symposium (USENIX Security 20)*. 2003–2020.

[79] Andrew C Yao. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 160–164.

[80] Mao Ye, Chengyue Gong, Lizhen Nie, Denny Zhou, Adam Klivans, and Qiang Liu. 2020. Good subnetworks provably exist: Pruning via greedy forward selection. In *International Conference on Machine Learning*. PMLR, 10820–10830.

[81] Xinche Zhang. 2021. The AlexNet, LeNet-5 and VGG NET applied to CIFAR-10. In *International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*. 414–419. https://doi.org/10.1109/ICBASE53849.2021.00083

[82] Chuan Zhao, Shengnan Zhao, Minghao Zhao, Zhenxiang Chen, Chong-Zhi Gao, Hongwei Li, and Yu-an Tan. 2019. Secure multi-party computation: theory, practice and applications. *Information Sciences* 476 (2019), 357–372.

[83] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. 2019. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems* 30, 11 (2019), 3212–3232.

A Glossary

The following table summarizes the notations used in the paper.

Table 8: Notations used in the paper.

Symbol	Notation
A_o	The classification accuracy of the original deep learning model. First defined in Section 4.1.
A_s	The classification accuracy for authorized clients, who can get decrypted FHE neuron outputs (demonstrated in Figure 3 (b)). First defined in Section 4.3.
A_r	The classification accuracy for unauthorized clients who cannot obtain decrypted FHE neuron outputs (Figure 3 (c)). First defined in Section 4.3.
A_{rec}	The classification accuracy achieved by the curious clients in a model recovery attack. First defined in Section 6.1.
A_t	The classification accuracy for models trained from scratch with 1000 samples. First defined in Section 6.1.
s	The quality of the core set selection. First defined in Section 4.3, Equation 1.
T	Model execution time for a batch of input samples.
$n_{i,1}$	The number of core neurons (with encrypted weights) in the ith convolution layer. Defined in Section 4.4.
$n_{i,2}$	The number of non-core neurons (with plaintext weights) in the ith convolution layer. Defined in Section 4.4.
N_e	The total number of core neurons (with encrypted weights). First defined in Section 4.1.
N	The total number of convolution neurons in the DNN. First defined in Section 4.1.
ER	The DNN encryption ratio, i.e., the proportion of core neurons out of all convolution neurons in the DNN. First defined in Section 5.3.

B Sample Recovery Attack

We adopt two mechanisms, split learning [23] and autoencoder [11], to recover the raw pixels of the input images from the output of the intermediate layers of the network. We evaluate an extreme case where all the outputs from layer 2 are known to the attacker, i.e., the curious model owner. Note that such outputs are obfuscated (multiplied by a random value ϵ). To fully mimic the server's capacity that she has the full training samples, we assume the server will attempt to rescale the obfuscated intermediate results, based on her knowledge of her own training samples, to break the client's obfuscation.

In Figure 8, we show the original input images (first row) and the recovery attack results of split learning (second row). As shown, the malicious model owner could not recover any meaningful image.

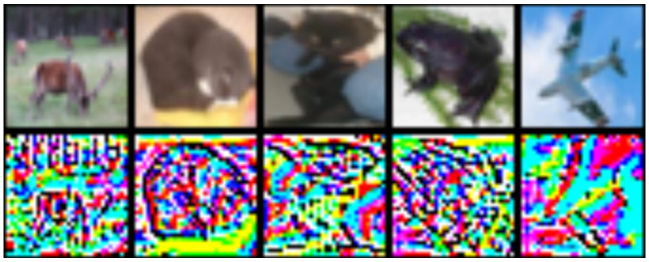


Figure 8: The model owner's sample recovery attack with split learning [23] when selecting 100% neurons.

In Figure 9, we show the original input images (first row), images recovered by the autoencoder (second row) from unobfuscated intermediate results, and the recovery attack results of the autoencoder (third row) from obfuscated intermediate results. Even with a well-trained autoencoder, the server cannot recover any meaningful image due to the protection of obfuscation. We also like to note that, in real-world attacks, the model owner is unlikely to have the entire layer 2 encrypted, hence, her capability is even weaker than the example attacks in this section.

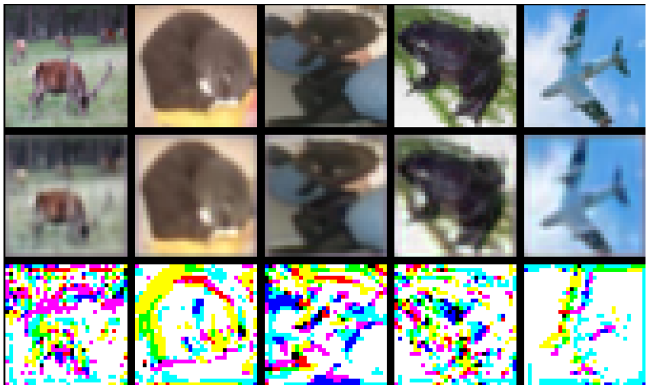


Figure 9: The model owner's sample recovery attack with an autoencoder [11] when selecting 100% neurons.

C Datasets and Model Structures

We adopt five popular benchmarking datasets for machine learning applications, as summarized in Table 9. They are briefly introduced as follows:

Table 9: Datasets and models used in the experiments.

	M	E	G	C	T
Complexity	Low	Medium	Medium	High	High
Image size	28×28	28×28	32×32	32×32	64×64
Categories	10	26	43	10	200
Train size	60000	124800	39209	50000	100000
Test size	5000	10400	6315	5000	10000
DNN Model	LeNet-5	LeNet-5	AlexNet	VGG16	ResNet18
Optimizer	Adam	Adam	SGD	SGD	SGD
Learning rate	1e-3	3e-3	5e-2	5e-2	5e-2
Scheduler	CosineAnnealingLR				
Accuracy(A_o)	99.36%	93.08%	93.51%	90.22%	72.00%

1. M: MNIST [18], E: EMNIST [14], G: GTSRB [38], C: CIFAR10 [47], T: Tiny-ImageNet [59] **2.** We adopt top-5 accuracy for Tiny-ImageNet.

Table 10: The structure of the LeNet-5 classifier for MNIST.

Layer	Type	Output	Kernel	Padding	Activation
1	Conv	6	5×5	0	-
2	AvePool	6	2×2	-	Square
3	Conv	16	5×5	0	-
4	AvePool	16	2×2	-	Square
5	FC	120	-	-	ReLU/Square
6	FC	84	-	-	ReLU/Square
7	Softmax	10	-	-	-

Table 11: The structure of the LeNet-5 classifier for EMNIST.

Layer	Type	Output	Kernel	Padding	Activation
1	Conv	10	5×5	0	-
2	AvePool	10	2×2	-	Square
3	Conv	20	5×5	0	-
4	AvePool	20	2×2	-	Square
5	FC	120	-	-	ReLU
6	FC	84	-	-	ReLU
7	Softmax	27	-	-	-

- **MNIST**. The MNIST dataset [18] contains grayscale images of handwritten digits from 0 to 9. It is a classic dataset for handwritten character recognition. MNIST was derived from a larger dataset, NIST Special DB 19, which also contains uppercase and lowercase letters. We adopt the classic LeNet-5 classifier [49] for MNIST.
- **EMNIST**. The Extended MNIST (EMNIST) [14] is a NIST variant for challenging classification tasks while sharing the same image structure and parameters as MNIST. We use the EMNIST Letters dataset, which contains 26 balanced classes of English letters. To get better accuracy, we modify LeNet-5 by increasing the first layer channels from 6 to 10 and the second layer channels from 16 to 20.
- **GTSRB**. The German Traffic Sign Recognition Benchmark [38] is an RGB dataset with 43 different types of traffic signs. We resize all the images to 32×32 pixels in size. We adopt the AlexNet model [48] and follow the solution in [81] to reduce the kernel size to 5 and 3 for the first two layers, respectively, to fit GTSRB.
- **CIFAR10**. The Canadian Institute For Advanced Research 10 (CIFAR10) dataset [47] is an RGB image dataset with 10 classes,

Table 12: The structure of the modified AlexNet for the GTSRB dataset.

Layer	Type	Output	Kernel	Padding	Activation
1	Conv	96	5×5	1	-
2	AvePool	96	2×2	-	Square
3	Conv	256	3×3	1	-
4	BN	256	-	-	-
5	MaxPool	256	2×2	-	ReLU
6	Conv	384	3×3	1	ReLU
7	Conv	384	3×3	1	ReLU
8	Conv	256	3×3	1	-
9	MaxPool	256	2×2	-	ReLU
10	FC	512	-	-	-
11	Dropout	512	-	-	ReLU
12	FC	128	-	-	-
13	Dropout	128	-	-	ReLU
14	Softmax	27	-	-	-

such as automobile, bird, cat, etc. We adopt a more complex model, VGG16 [75], for CIFAR10.

• **Tiny-ImageNet**. Tiny-ImageNet [59] is a modified subset of the original ImageNet [17]. It contains 200 different classes of 64×64 colored images. We adopt ResNet18 [32] to classify it.

Next, we present the detailed architectures of the deep learning models used in the experiments: the two LeNet-5 networks for MNIST and EMNIST, respectively, the AlexNet network for GTSRB, the VGG16 network for CIFAR10, and the ResNet18 for Tiny-ImageNet.

Table 13: The structure of the VGG16 model used for CIFAR10

Layer	Type	Output	Kernel	Padding	Activation
1	Conv	64	3×3	1	Square
2	Conv	64	3×3	1	-
3	BN	64	-	-	-
4	MaxPool	64	2×2	-	ReLU
5	Conv	128	3×3	1	-
6	BN	128	-	-	ReLU
7	Conv	128	3×3	1	-
8	BN	128	-	-	-
9	MaxPool	128	2×2	-	ReLU
10	Conv	256	3×3	1	-
11	BN	256	-	-	ReLU
12	Conv	256	3×3	1	-
13	BN	256	-	-	ReLU
14	Conv	256	3×3	1	-
15	BN	256	-	-	-
16	MaxPool	256	2×2	-	ReLU
17	Conv	512	3×3	1	-
18	BN	512	-	-	ReLU
19	Conv	512	3×3	1	-
20	BN	512	-	-	ReLU
21	Conv	512	3×3	1	-
22	BN	512	-	-	-
23	MaxPool	512	2×2	-	ReLU
24	Conv	512	3×3	1	-
25	BN	512	-	-	ReLU
26	Conv	512	3×3	1	-
27	BN	512	-	-	ReLU
28	Conv	512	3×3	1	-
29	BN	512	-	-	-
30	MaxPool	512	2×2	-	ReLU
31	FC	512	-	-	-
32	Dropout	512	-	-	ReLU
33	FC	512	-	-	-
34	Dropout	512	-	-	ReLU
35	Softmax	10	-	-	-

Table 14: The structure of the ResNet18 model used for Tiny-ImageNet

Layer	Type	Output	Kernel	Padding	Activation
1	Conv	64	3×3	1	-
2	BN	64	-	-	Square
3	Conv	64	3×3	1	-
4	BN	64	-	-	ReLU
5	Conv	64	3×3	1	-
6	BN	64	-	-	ReLU
7	Shortcut	64	-	-	-
8	Conv	128	3×3	1	-
9	BN	128	-	-	ReLU
10	Conv	128	3×3	1	-
11	BN	128	-	-	
12	Shortcut	128	-	-	-
13	Conv	128	3×3	1	-
14	BN	128	-	-	ReLU
15	Conv	128	3×3	1	-
16	BN	128	-	-	
17	Conv	256	3×3	1	-
18	BN	256	-	-	ReLU
19	Conv	256	3×3	1	-
20	BN	256	-	-	
21	Shortcut	256	-	-	-
22	Conv	256	3×3	1	-
23	BN	256	-	-	ReLU
24	Conv	256	3×3	1	-
25	BN	256	-	-	
26	Conv	512	3×3	1	-
27	BN	512	-	-	ReLU
28	Conv	512	3×3	1	-
29	BN	512	-	-	
30	Shortcut	512	-	-	-
31	Conv	512	3×3	1	-
32	BN	512	-	-	ReLU
33	Conv	512	3×3	1	-
34	BN	512	-	-	
35	Softmax	200	-	-	-